AD-A272 322

(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DTIC
ELECTE
NOV 0 5 1993
S B D

THESIS

A Software Architecture for a Small Autonomous
Underwater Vehicle Navigation System

by

Clark D. Stevens

June, 1993

Primary Advisor: Se-Hung Kwak

Approved for public release; distribution is unlimited.

93-27108

8

Unclassified
Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification: Unclassified		1b Restrictive Markings		
2a Security Classification Authority		3 Distribution/Availability of Report Approved for public release; distribution is unlimited.		
2b Declassification/Downgrading Schedule				
4 Performing Organization Report Number(s)		5 Monitoring Organization Report Number(s)		
6a Name of Performing Organization Naval Postgraduate School	6b Office Symbol *(if applicable)* 37	7a Name of Monitoring Organization Naval Postgraduate School		
6c Address *(city, state, and ZIP code)* Monterey CA 93943-5000		7b Address *(city, state, and ZIP code)* Monterey CA 93943-5000		
8a Name of Funding/Sponsoring Organization	6b Office Symbol *(if applicable)*	9 Procurement Instrument Identification Number		
Address *(city, state, and ZIP code)*		10 Source of Funding Numbers		
		Program Element No	Project No	Task No / Work Unit Accession No
11 Title *(include security classification)* A SOFTWARE ARCHITECTURE FOR A SMALL AUTONOMOUS UNDERWATER VEHICLE NAVIGATION SYSTEM				
12 Personal Author(s) Clark D. Stevens, LCDR, USN				
13a Type of Report Master's Thesis	13b Time Covered From09/92 To06/93	14 Date of Report *(year, month, day)* 1993, JUNE	15 Page Count 154	
16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
17 Cosati Codes: Field / Group / Subgroup	18 Subject Terms *(continue on reverse if necessary and identify by block number)* AUTONOMOUS UNDERWATER VEHICLE (AUV), GLOBAL POSITIONING SYSTEM (GPS), INERTIAL NAVIGATION SYSTEM (INS)			

19 Abstract *(continue on reverse if necessary and identify by block number)*

This thesis documents the development of an interim Small Autonomous Underwater Vehicle (AUV) Navigation System (SANS), a self-contained, externally mountable navigation package. The purpose of SANS is to determine the position of a submerged object of interest located by an AUV. The volume of SANS must not exceed 120 cubic inches and total system accuracy of 10.0 meters rms or better is required. An Inertial Navigation System (INS) is implemented to compute the ascent path during transit from an object of interest to the surface. INS hardware components include miniature spin gyroscopes, a compass and a depth transducer interfaced through an analog to digital converter. Global Positioning System (GPS) is used to determine the AUV's location after reaching the surface. The reciprocal of the ascent vector is applied to the AUV's GPS position to accurately determine the location of the target of interest. A primarily object-oriented software architecture is implemented here with extensive software testing to verify the proper operation of key modules.

The objective of this thesis is to quantify the adequacy of the selected components in meeting these requirements and to develop a breadboard design demonstrating the basic functions of the interim SANS. This research conclude that the components selected for the interim SANS meet the accuracy requirements provided the AUV maintains a climb angle which is equal to or steeper than 12 degrees from a typical mission depth of 20 meters.

20 Distribution/Availability of Abstract __ unclassified/unlimited __ same as report __ DTIC users	21 Abstract Security Classification Unclassified	
22a Name of Responsible Individual Dr. Se-Hung Kwak	22b Telephone *(include Area Code)* (408) 646-2168	22c Office Symbol CS/KW

DD FORM 1473,84 MAR — 83 APR edition may be used until exhausted — security classification of this page

All other editions are obsolete — Unclassified

Approved for public release; distribution is unlimited.

A Software Architecture for a Small Autonomous Underwater Vehicle Navigation System

by

Clark D. Stevens
Lieutenant Commander, United States Navy
B.S., University of South Carolina, 1979

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1993

Author: Clark D. Stevens
Clark D. Stevens

Approved by: Se-Hung Kwak
Se-Hung Kwak, Primary Advisor

James R. Clynch
James R. Clynch, Associate Advisor

Gary Hughes
Cdr. Gary L. Hughes, USN, Chairman
Department of Computer Science

ABSTRACT

This thesis documents the development of an interim Small Autonomous Underwater Vehicle (AUV) Navigation System (SANS), a self-contained, externally mountable navigation package. The purpose of SANS is to determine the position of a submerged object of interest located by an AUV. The volume of SANS must not exceed 120 cubic inches and total system accuracy of 10.0 meters rms or better is required. An Inertial Navigation System (INS) is implemented to compute the ascent path during transit from an object of interest to the surface. INS hardware components include miniature spin gyroscopes, a compass and a depth transducer interfaced through an analog to digital converter. Global Positioning System (GPS) is used to determine the AUV's location after reaching the surface. The reciprocal of the ascent vector is applied to the AUV's GPS position to accurately determine the location of the target of interest. A primarily object-oriented software architecture is implemented here with extensive software testing to verify the proper operation of key modules.

The objective of this thesis is to quantify the adequacy of the selected components in meeting these requirements and to develop a breadboard design demonstrating the basic functions of the interim SANS. This research concludes that the components selected for the interim SANS meet the accuracy requirements provided the AUV maintains a climb angle which is equal to or steeper than 12 degrees from a typical mission depth of 20 meters.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

The author wishes to express his admiration and appreciation to Dr. Se-Hung Kwak who served as primary advisor in this research. His broad base of experience contributed immeasurably to the learning process. Dr. Kwak helped me to "understand" more about computer hardware during the design of the gyroscope driver described in the text than I have "learned" in hundreds of hours of instruction. Most of all, his contagious enthusiasm and eternally cheerful smile spurred me on in the darkest of hours.

A debt of gratitude is certainly owed to Dr. James Clynch, who served as second reader. His careful guidance and wisdom broadened the author's appreciation for the concepts involved in systems engineering and scientific research. His willingness to spend long hours in assisting the author, often at short notice, contributed immeasurably to the quality of the finished product while greatly relieving the stress involved.

Dr. Robert McGhee has been a great pleasure to work with and to know. His steady guidance as principal investigator has and will most certainly continue to make valuable learning experiences such as this available to many students.

Mr. Russ Whalen provided the technical knowledge which made this entire research practical. Above all else, his friendship provided a sanity check when needed most.

Last, but certainly not least, the author wishes to thank his wife, Cathy, and his three wonderful children, Laura, Greg, and Alyssa. They have selflessly given the understanding and support which made the successful completion of this work possible.

I. INTRODUCTION

A. BACKGROUND

Many of the missions proposed for Autonomous Underwater Vehicles (AUV's) require a high degree accuracy in navigation. The Global Positioning System (GPS) provides perhaps the most accurate navigation information currently available. The feasibility of incorporating GPS into a Small AUV Navigation System (SANS) was evaluated by the thesis of James Bernard McKeon with a favorable assessment. Some major concerns addressed by McKeon were accuracy, acquisition time (to limit the probability of counter-detection) and power consumption. [Ref. 1]

B. PURPOSE

There are two distinct phases embodied in the eventual employment of a SANS. The first of these is the transit phase involving navigation of the AUV from a launch position to an area where an underwater mapping mission is to be conducted. After the mapping mission is complete, the AUV will transit back to a recovery position. This mission may involve waypoint steering and artificial intelligence applications such as obstacle avoidance. The second distinct phase called the mission phase involves execution of the mapping mission. The objective here is to determine the locations of underwater objects identified by the AUV in the area of interest.

The purpose of this thesis is to document the development of an interim system meeting the mission execution (mapping phase) objectives of the SANS. Specifically,

this interim system will occupy less than 120 cubic inches and will demonstrate the feasibility of determining the position of a submerged object located by an AUV to within 10 meters. The term AUV should include any small underwater vehicle which could easily carry such a compact device.

C. PRINCIPLES OF OPERATION

The interim system seeks to incorporate GPS position fixing into the SANS. In order to resolve the location of a submerged object, two problems must be solved. First, the path travelled from the submerged object to the surface must be determined since GPS signals cannot penetrate water. Secondly, the AUV's position on the surface must be accurately determined so that the location of a submerged object may be resolved by applying the reciprocal of the ascent path.

Commercially available inertial navigation systems will soon exist which exceed the accuracy requirements and are compatible with the volume requirements of SANS. This would permit identification of multiple targets with only occasional GPS updates for navigation accuracy. This capability will be explored in future research. In the interim SANS, a pop-up mode is employed in order to locate each underwater object. The AUV's ascent path is calculated trigonometrically by measuring the depth change, pitch angle and compass heading throughout the ascent. On the surface, data from the GPS receiver is recorded to resolve the location of the target through post-processing.

D. SCOPE OF THESIS

This thesis reports the findings of the second year of research in an ongoing research project. The objectives of this thesis are limited to defining the hardware and software architecture required to conduct the mapping phase of the mission. These objectives are described by McGhee et al. and include [Ref. 2] :

a. Low Power Consumption. Operation from an external battery pack for 24 hours is desirable.

b. GPS antenna exposure time in survey area should be minimized. Up to 30 seconds exposure allowed but intervals between such exposures should be as long as possible, exceeding several minutes at a minimum.

c. The GPS antenna should present a very small cross section when exposed and should not extend more than a few inches above the surface.

d. For the mapping phase of the mission, system positioning accuracy of 10 meters rms or better is required with post-processing, submerged as well as surfaced.

e. Total volume not to exceed 120 inches. Elongated, streamlined packaging is preferred.

The feasibility of meeting the objectives of the mapping phase is demonstrated initially by a "breadboard" system design. The performance of individual components is assessed through analysis of data collected from the "breadboard" system operating onboard an instrumented golf-cart traversing a surveyed test track. Eventually, a "wet design" prototype is planned which will be carried externally aboard the Naval Postgraduate School AUV-2 as part of a technology demonstration. The transit and

return phases characterized by more relaxed navigation accuracy requirements will be addressed in future research.

E. THESIS ORGANIZATION

This thesis explores hardware components required to implement the interim SANS and the software architecture to support this mission execution. While the technical expertise of the author lies more in the realm of the software design, both topics are covered in sufficient detail to ensure that the overall system performance satisfies the design objectives.

Chapter II reviews previous and ongoing work, especially that which relates to the incorporation of GPS and inertial navigation into AUV navigation.

In Chapter III, a detailed problem statement is presented. The requirements for the GPS serial line interface and translation of the binary format is examined. Problems related to the computation of the distance travelled from datum are discussed. Possible solutions involving gyros or accelerometers to determine the climb angle are examined.

Chapter IV details the design of the SANS. Those characteristics of the individual hardware components which are significant to the design of the SANS are presented. The software design methodology and philosophy is presented in this chapter.

Chapter V delineates the process and results of software testing. Various approaches to testing are considered and a testing philosophy and strategy are developed. Reliability of key modules are examined.

In Chapter VI, the experimental data gathered during system validation is presented along with detailed analysis. The accuracy and adequacy of individual components is described with comparison to advertised technical specifications. The performance of the complete system is also assessed.

Chapter VII concludes with a summary of the results of this research. Future research topics are also described.

II. SURVEY OF PREVIOUS WORK

A. AUV NAVIGATION

As stated by McKeon [ref. 1]:

> Approaches to navigating of AUV's fall into two different categories: sensor based navigation and external signal based navigation. Sensor based navigation refers to an AUV navigational system that is self contained.

For AUV missions of extended duration (in excess of 100 hours), current sensor based navigation systems alone are considered incapable of providing the accuracy necessary to perform precise mapping operations. Most external signal navigation systems require continuous exposure of some sort of signal receiver. This is a serious restriction which negates many of the advantages of an AUV. McKeon proposes a navigation system combining the strengths of sensor based and external signal based navigation systems for AUV navigation. [Ref. 1]

Previously, the most widely available forms of external signal navigation systems were Loran and Omega. Both of these are considered unsuitable for AUV navigation due to limited coverage, relative inaccuracy and the requirement for uninterrupted signal reception in Loran.

B. INCORPORATION OF GPS INTO AN AUV

With the Initial Operational Capability (IOC) of the GPS satellite network in 1993, continuous worldwide GPS coverage will be available [Ref. 3]. This combined with the accuracy of GPS shown in Table 2.1 makes it an ideal candidate for use as an

TABLE 2.1 : GPS POSITIONING ACCURACY
(IN METERS)

	PPS	SPS
NON-DIFFERENTIAL	16	100
DIFFERENTIAL	2-4	2-4

external signal navigation source in AUV's. McKeon provides a detailed analysis of the expected accuracy of an AUV based GPS receiver with consideration for the requirement to minimize antenna exposure. A navigation solution was achieved in 30 seconds or less with a 57% success rate (86 of 150 cases). The position errors fell within the expected range with a standard deviation of horizontal error of 29 meters. [Ref. 1]

Dr. SeHung Kwak developed an assembly language serial line interface for a GPS receiver during implementation of a Suitcase Navigation Data Logger (SNDL) [Ref. 4]. This interface is used extensively in the implementation of the GPS and AUV communications here because of its compatibility with Ada, the development language for SANS.

C. INERTIAL NAVIGATION IN AUV'S

In Reference 2, McGhee et al. examine the incorporation of available inertial navigation systems into an AUV. These systems include fiber optic gyros, ring laser gyros and vibratory rate sensors. Analysis shows that systems exceeding the accuracy

requirements within the size and power consumption restrictions outlined in Chapter 1 are feasible and should be available in 1994. These findings are supported by the work of Hutchinson et al. in Reference 5.

The utilization of GPS to correct inertial measurements for navigation of an AUV is addressed by Brown [Ref. 6] and Nagengast [Ref. 7]. A model based on a medium grade INS using ring laser gyro's (RLG's) and combinations of high and low accuracy GPS was used to simulate actual conditions. The results from computer simulations matched expectations with one nautical mile per hour drift in the INS. Nagengast's estimate of GPS error is 43 meters (one standard deviation of horizontal error). This is larger than McKeon's result but still within the expected range.

In Reference 8, Miller develops an extended Kalman filter adapted for an AUV which seeks to optimize the INS navigation solution.

D. KALMAN FILTERING TECHNIQUES

A major objective in the incorporation of GPS into an AUV, especially in a tactical environment, is the requirement to minimize the probability of detection. The SANS must optimize the accuracy of the GPS position while minimizing the duration of antenna exposure. No previous work has been identified which addresses the incorporation of a Kalman filtering technique to update a navigation solution for an AUV using GPS data. This is due to the narrow scope of the problem in AUV navigation where antenna exposure must be minimized and time between exposures

may be large. Most commercial GPS receivers already incorporate some filtering technique internally to produce position fixes.

III. DETAILED PROBLEM STATEMENT

A. GPS NAVIGATION

"The GPS has two levels of accuracy, the Standard Positioning Service (SPS), and the Precise Positioning Service (PPS)." [Ref. 3] In SPS, an intentional inaccuracy is introduced into the satellite broadcast signals to degrade the accuracy of non-PPS receivers through a process called Selective Availability (SA). This limits the accuracy of the GPS solution of non-PPS receivers in real-time to 100 meters (two standard deviations horizontal error) unless differential processing is used. PPS processing requires the use of cryptographic keys to decode and remove the error in broadcast signals and yields an accuracy of 16 meters as shown in Table 2.1. [Ref. 10]

The use of cryptographic keys aboard an AUV is undesirable. Even with relatively tamper-proof secure devices, the potential for loss causes concern. As seen in Table 2.1, real-time differential processing or differential post-processing can improve GPS accuracy to 2-4 meters (SA on or off) negating any advantage of processing with SA off. This also alleviates the concern of placing cryptographic keys aboard an unmanned vehicle.

A commercial GPS receiver manufactured by Motorola is being used to conduct research in the development of the interim SANS system. In order to achieve the objectives for accuracy during the mapping phase established in Section I.D.d, differential processing is required. To permit differential post-processing, the GPS

receiver transmits satellite range format messages containing "raw" (unprocessed) satellite range data. These messages are transmitted through a serial connection to the host computer where they are stored in non-volatile memory. A second receiver operating concurrently at a known location is used to determine and record the total error degrading GPS accuracy. This error includes inaccuracy due to atmospheric conditions and SA. The error in the range data recorded by the SANS is corrected during post-processing to yield the accuracy described in Table 2.1.

Position format messages, which are another message format transmitted by the GPS receiver, permit navigation updates during mission execution or during transit. Both position format and satellite range format messages are transmitted in Motorola proprietary binary format in RS-232 compatible serial stream. In position format messages, the latitude and longitude are 32-bit two's complement encoded in 4 bytes each. This binary format necessitates the development of a module capable of receiving, storing, and processing variable length binary data streams.

In order to conserve power, the GPS receiver must be unpowered when not in operation. Since this receiver cannot output position and satellite range format messages at the same time, initialization must provide for transmission of the appropriate message format depending on the current mission phase. During transit phase, the receiver must be initialized to transmit position format messages for position updates. During mapping phase, satellite range format message transmission must be initialized to provide satellite range data for post-processing.

B. SUBMERGED NAVIGATION

In order to calculate the distance from a submerged object to the surface where an accurate GPS fix can be obtained, the depth change, climb angle and direction of travel (heading) must be known. Heading is measured by a compass and depth change is measured by a depth transducer. The compass and depth transducer selected for use in SANS both produce an analog output which is converted to digital data by an Analog to Digital (A to D) converter. The specifications and interface for these are covered in the section IV.A (system hardware) and Appendix A (technical specifications).

Figure 3.1 and equation 3.1 illustrate the trigonometry involved in computation of horizontal distance. Figure 3.2 and equations 3.2 and 3.3 illustrate how heading and horizontal distance travelled are resolved into latitude and longitude components. Rather than computing a single vector travelled at the surface, SANS sums a series of vectors calculated for small increments of time during the ascent. This should improve accuracy due to frequent heading and climb angle changes through the ascent. A differential error analysis is conducted using this approach in section VI.D.2. In computing the horizontal distance travelled in equation 3.1, as the climb angle approaches zero, the horizontal distance travelled will approach infinity (the AUV will never reach the surface since the climb angle is zero). Therefore, the minimum climb angle consistent with system accuracy objectives for the interim SANS system must be quantified.

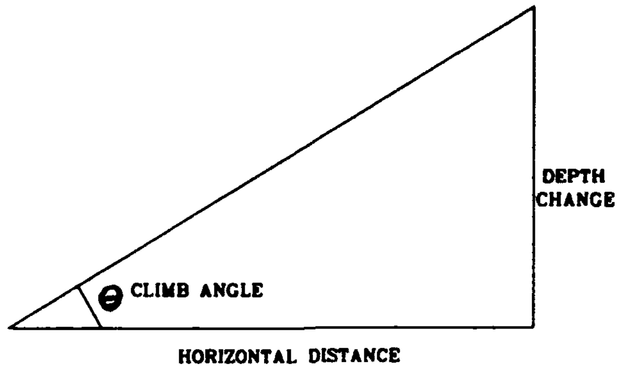


Figure 3.1 : Computation of Horizontal Distance Travelled

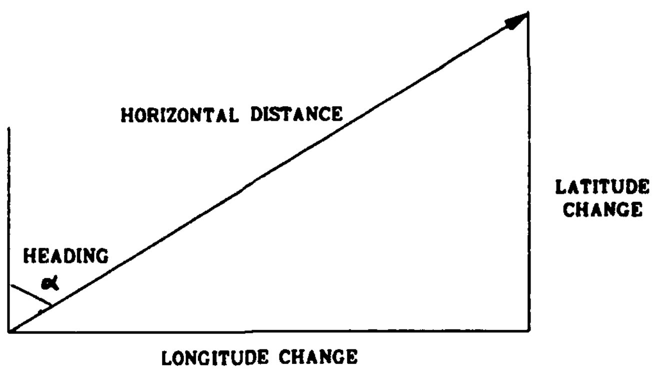


Figure 3.2 : Computation of Latitude/Longitude Change

$$s_H = \frac{DEPTHCHANGE}{TAN\theta} \tag{3.1}$$

$$\delta\phi = s_H * COS\alpha \tag{3.2}$$

$$\delta\lambda = s_H * SIN\alpha \tag{3.3}$$

Where θ is the climb angle.

$\varnothing$ is the latitude

lambda is the longitude

The measurement of the climb angle presents the greatest challenge within the space requirements of the SANS. Two approaches are developed here using a miniature gyroscope (gyro) and a single axis accelerometer.

1. Gyroscope

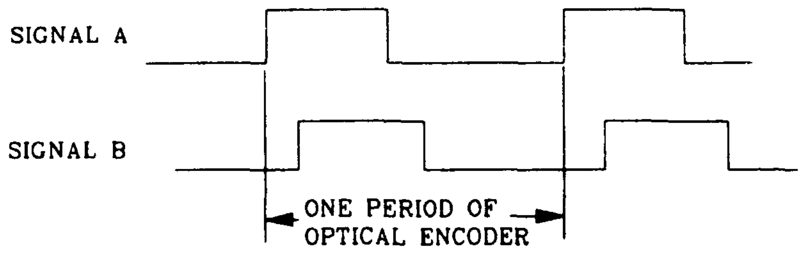
A miniature spin gyro manufactured by Gyration Inc. is small enough for use in SANS and has a very low power requirement (0.1 watts). While the drift rate of the gyro is high, it should be acceptable for the short period of time required to travel from a submerged object to the surface. The error rate of the gyro is evaluated and

impact on system accuracy are assessed in section VI.D along with other system hardware.

The gyro's rotational positional information is encoded in two phase and quadrature digital data signals as illustrated in Figure 3.3. These signals encode the output of an optical encoder. The direction of gyro rotation can be determined by observing which signal rises first and the amount of rotation can be determined by counting the number of pulses. One period of rotation in the optical encoder begins at the rising edge of a pulse in either output signal. The period includes both pulses in the pulse pair and ends at the beginning of the next rising edge in either output signal. One period represents 0.8 degrees of rotation in a low-resolution gyro (0.4 degrees in high-resolution). Only relative position is available from the gyro. The solution in SANS assumes a near level attitude for the AUV at the time an object of interest is located on the bottom. Climb angles during the ascent are relative to this assumed horizontal attitude.

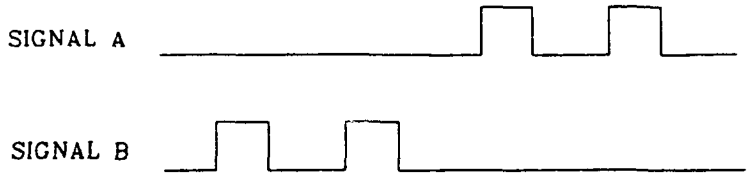
The manufacturer provides a development test box which converts this output into a serial data stream. This test box is too bulky for inclusion in SANS so use of gyro information necessitates development of a module to track gyro rotation. This rotation can be applied to the last known climb angle at regular intervals during the ascent to determine the gyro's new relative position.

A significant problem encountered by Gyration Inc. is high frequency noise in the output signal while the gyro is stationary as shown in Figure 3.4. This noise is



*NOTE : Overlapping pulses indicates valid gyro rotation

Figure 3.3 : Phase and Quadrature Signal Output



Note : Non-overlapping pulses indicate presence of noise only
(no valid gyro rotation is present)

Figure 3.4 : Noise in Phase and Quadrature Signal Output

random and can occur in one or both channels. The software solution developed to track gyro movement must filter out this noise. This can be accomplished by taking advantage of the fact that signals describing valid rotation must follow certain state transition patterns. These patterns will be used in the next chapter to develop a state transition algorithm to count valid state transitions while rejecting invalid state transitions as noise. The pulse count can then be incremented or decremented by the algorithm accordingly.

2. Accelerometer

A second approach which may be developed to determine climb angle in the SANS is inertial measurement based on an accelerometer. McGhee develops a finite approximation of climb angle using the output of a single-axis accelerometer. This is accomplished by sensing the component of gravity along the axis measured by the accelerometer. The longitudinal acceleration of the AUV is assumed to be negligible (constant velocity). This allows the x-axis (vertical) acceleration to be used to determine the climb angle. [Ref. 9]

McGhee points out that a measurement of rotation based on accelerometers will yield best results during low frequency rotation since the precession error in the gyro is predominant. The reverse is true during high frequency pitch attitude excursions where gyros are preferred since the effect of precession error is minimized. A hybrid solution using both accelerometers and gyros would optimize the strengths of each approach. [Ref. 9]

Similar logic applies in measuring heading change. The compass is preferred under stable conditions due to the predominance of precession error in the gyro. During a high rate of rotation, however, the directional gyro yields a better result since the effect of precession error is minimized. Again, a hybrid solution using the gyro to smooth compass heading changes during turns is preferred.

IV. SYSTEM DESIGN

A. HARDWARE DESCRIPTION

A brief description of each hardware device is provided here to summarize some of the significant characteristics essential to the design of the interim SANS. Appendix A provides a more detailed summary of each device's technical specifications. Figure 4.1 illustrates the hardware interface.

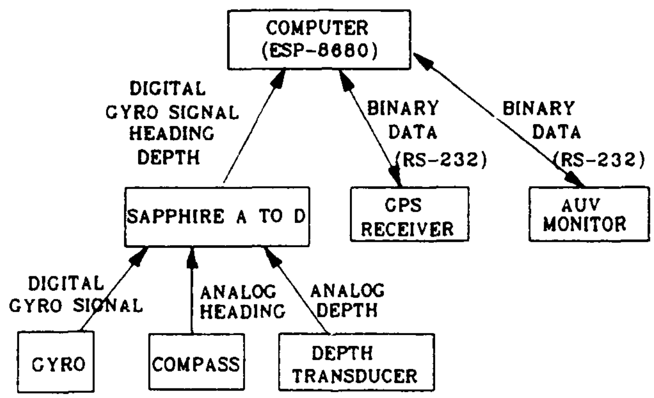


Figure 4.1 : SANS Hardware Devices Interface

1. Sapphire Converter

The Diamond Systems Sapphire card provides a variety of features. The key features utilized in the development of SANS are the Analog to Digital (A to D) conversion and the Digital to Digital (D to D) Input/Output (I/O) capability.

The Sapphire D to D interface provides a 7-bit digital I/O port through a DB-37 connector (37-pin). These include 3 input and 4 output pins. The DB-37 also provides a single external trigger input channel. A 24-bit general purpose I/O port is also provided through the 82C55 programmable interface. Each group of 12 pins may be programmed in sets of 4 pins and 8 pins as inputs or outputs. This permits definition of three 8-bit ports (A through C). Port C can be divided into 2 4-bit ports under mode control. Each of these 4-bit ports can be combined with an 8-bit port to provide 12-bit input/output if desired.

The analog interface provides 8 analog input channels through the software controlled input channel multiplexer. While there are 3 selectable voltage ranges (+/- 5 volts, 0-10 volts and +/- 10 volts), the multiplexer can actually tolerate voltages of up to +/- 32 volts. 8-bit output resolution (1 part in 256) or 12-bit output resolution (1 part in 4096) with a 20 micro-second A to D conversion time is software selectable for each analog input channel. [Ref. 10]

One serious limitation imposed by the Sapphire converter is the availibilty of only a single interrupt input channel. This restricts the current software architecture to tracking a single gyro at any given time. The interface chosen for the target machine should have at least two interrupt channels so that the directional gyro may be used to aid in heading updates. This is especially important during high rates of turn as discussed in Section III.B.2.

2. Core Module

The Dover Electronics ESP 8680 (Extra Small Package) core module provides an 8086 equivalent central processor. One megabyte Dynamic Random Access Memory (DRAM) is provided as well as two serial ports (RS-232). This module was chosen for its compactness (1.7 inches by 5.2 inches by 1/3 inch) and for its power conservation features. Several modes of operation are provided including full speed operation at 14 megahertz and a low power consumption "drowsy mode" operating at 1 megahertz. Typical power consumption is 0.2 watts at 8 megahertz [Ref. 11]. Two megabyte EPROM (Eraseable Programmable Read-Only Memory) non-volatile memory modules are currently available. 20 megabyte memory modules are expected to be available shortly. Additionally, the manufacturer is currently developing a miniaturized A to D converter with capabilities equivalent to the Sapphire A to D converter.

3. GPS Receiver

The Motorola PVT6 is a lightweight miniature GPS receiver capable of tracking six satellites simultaneously. The receiver is capable of real-time differential processing and can be upgraded to support PPS. Acquisition time is 24 seconds typical time to first fix with an accuracy of 100 meters with SA on without differential processing. With differential processing, 1 - 5 meters accuracy is probable.

When power is applied, the receiver is initialized in the mode that existed when power was last removed. The most recent valid coordinates are stored by the receiver

through reinitialization. This is accomplished by transmitting a formatted binary command (message) through the serial communication line.

Serial stream output is provided through an RS-232 standard interface in a variety of formats. These include National Marine Electronics Association (NMEA) standard format, Motorola Proprietary Binary format and LORAN input/output format. Although the NMEA standard format would be very attractive from the standpoint of compatibility between various receivers, the Motorola Proprietary Binary format was selected since it provides both satellite range information for differential post-processing and position format messages (although not concurrently).

4. Gyroscopes

The Gyration GyroEngine is an optically sensed miniature spin gyroscope (gyro). There are two varieties of gyro available from Gyration, vertical and directional. Both are used in this interim system. The vertical gyro inner gimbals encoder measures the pitch attitude of the AUV during ascent from the object of interest. Roll attitude measured by the outer gimbals is not of interest in this system. The directional gyro's outer gimbals reports yaw information and may be used to smooth the compass heading data during high turn rates.

The GyroEngine generates two phase and quadrature digital data signals from each gimbals encoder. A test box provided by Gyration, Inc. translates this signal into serial stream data interfaced through an RS-232 serial port connection. The test box is useful for evaluating the performance characteristics of the GyroEngine, but is too

bulky to be used in the SANS. A software solution has been developed to permit the GyroEngine to interface directly with the ESP-8680 through Sapphire A to D via Digital Input/Output (DIO). This solution is developed in Section IV.B.2.f and Appendix D.

5. Compass

The KVH C100 Multi-Purpose Digital Compass provides digital output in a 4 digit Binary Coded Decimal (BCD) serial stream format. Analog output is provided in the form of linear or sine/cosine voltage signal. Since the serial stream digital output would require an additional serial port connection in the ESP-8680, the compass analog output is interfaced through the Sapphire A to D converter. A linear voltage is produced proportional to heading in the range from 0.1 volts (000 degrees) through 1.9 volts (360 degrees). This voltage is converted to digital data by software triggered A to D conversion mode through the Sapphire A to D converter. The analog signal from the compass is connected to the Sapphire through one of 8 multiplexed input channels. This input channel is selectable in the parameters file A_TO_D.DAT at run-time.

6. Depth Transducer

The Omega Inc. PX176-100S5V Depth Transducer has an operating range of 0 to 100 pounds per square inch static pressure (PSIS). This equates to 6.7 standard atmospheres or 217 feet (67 meters) depth of sea water. Analog output is 1 to 6 volts direct current (DC). A to D conversion is performed by repetitive software triggered

single A to D conversions in the same manner as the compass signal with the analog input channel similarly selectable.

B. SOFTWARE ARCHITECTURE

The software design for the SANS, can be described at the highest level by three major operations. These are:

- Monitoring the AUV for a position fix request,

- Navigation data-logging for dead reckoning (DR) navigation (post-processed to determine ascent vector), and

- GPS data-logging for post-processed positional information.

AUV monitoring must be performed continuously in the event of a request by the AUV to determine the location of a submerged object of interest. This permits the GPS to be unpowered while the AUV is submerged and is intended as an energy conservation measure. DR navigation in the interim system consists of measuring the horizontal distance travelled from a submerged object to the surface as described in Figures 3.1 and 3.2. After the AUV reaches the surface, the GPS position is provided through a serial port connection from the Motorola GPS receiver.

The programming language used in the development of SANS is Meridian Ada version 4.1.1. Assembly language is used for low level, high frequency operations to improve efficiency. The object code for these assembly language modules are compiled by Borland's Turbo-Assembler and linked with the Ada object code using Borland's Turbo-Linker.

In order to produce a logical structure, a precise definition of design requirements is essential. "The primary product of this phase is an approved development (functional) software specification." as stated by Booch [Ref 11]. These requirements form the basis for development of the software structure. They also permit formulation of a test plan to evaluate the performance of the overall system and of individual modules.

a. AUV Monitoring.

AUV monitoring shall be performed continuously during the mapping phase so that when an object of interest has been located by the AUV, sufficient data may be recorded to determine the geographic position of the object. Communication with the AUV shall be via RS-232 serial line communication. DR navigation and GPS processing shall remain inactive with GPS unpowered (except for a low voltage applied to preserve static RAM (SRAM volatile memory) until the AUV requests a position fix.

b. GPS.

The GPS shall be initialized with the receiver in the Motorola proprietary binary format when the AUV requests a position fix. The receiver shall be programmed to transmit both position/status and satellite range format messages on a one second interval when initialized. The host computer shall be configured to receive RS-232 serial communications with connection parameters specified from an input data file at run-time. Upon mission termination, the host machine configuration shall

data file at run-time. Upon mission termination, the host machine configuration shall be restored to the state which existed prior to SANS execution. This is primarily to prevent side-effects due to improper configuration in the software development machine during software development.

All information received through the serial connection by the host machine shall be written to non-volatile memory prior to real-time processing. Satellite range messages will be recorded for post-processing only, while position/status information may be processed for navigation updates as necessary. Position format messages shall be processed to determine the current position and PDOP for navigation updates and the checksum to validate the particular message.

c. DR Navigation

DR navigation outputs shall include pitch attitude and heading in degrees and depth in meters. DR processing shall record input values with a precision commensurate with component accuracy and sufficient to satisfy system accuracy requirements specified in section I.D.d.

(1) Pitch Angle. Gyro position shall be monitored continuously by a process transparent to SANS operation. The process shall take a pair of standard phase and quadrature signals as inputs. Valid pulse pairs [Fig. 3.3] shall be counted on a continuous basis with each unit representing +/- 0.4 degrees of rotation depending on the direction of rotation. Invalid pulses from either signal [Fig. 3.4] are considered "noise" and shall be disregarded in the pulse count. The cumulative value of the pulse

count shall be continuously available for interrogation by the main procedure for DR navigation.

(2) A to D Conversions. Sapphire A to D conversions shall be initialized from a data file containing the multiplexed channels to be used for heading and depth inputs. Analog to digital conversion of analog heading and depth output shall provide 12 bit accuracy (one part in 4096). Digitized output shall be converted into meters for depth and degrees for heading.

2. Software Design

The object oriented design methodology described by Booch [Ref. 11] is used in the design of SANS. The object oriented approach was selected because it provides an advantage in the management of complexity of the system at the highest level over other design approaches.

As the system is decomposed hierarchically, some smaller elements are more logically approached using a functional design methodology. The functional design engineered by Dr. Kwak for the serial line interface was retained in SANS for both AUV monitoring and GPS processing. This approach is also well suited for the implementation of the procedure oriented assembly language driver for the gyro.

Booch suggests the following sequence in the object-oriented design approach [Ref. 11]:

- Identify the objects and their attributes

- Identify the operations that affect each object and the operations that each object must initiate

- Establish the visibility of each object in relation to other objects

- Establish the interface of each object

- Implement each object

a. Identify the Objects

The purpose of DR navigation in the interim system is to produce a vector travelled by the AUV from the submerged object to a position on the surface where a GPS fix is obtained. Figure 3.1 illustrates the computation of the horizontal distance travelled from the pitch angle and depth change during the ascent. Figure 3.2 describes the calculation of the change in latitude and longitude from the heading and horizontal distance travelled.

In order to accomplish the objectives of DR navigation, the AUV's heading, pitch angle and depth change must be accessed for each update cycle. This is accomplished repetitively in small increments throughout the ascent. These data items are treated as objects as described in Table 4.1. Every instance of each object type is stored in a file for post-processing. During post-processing, these values are used to calculate a heading and horizontal distance travelled by the AUV from the target located by the AUV to the surface. The reciprocal of this vector can be applied to the GPS position on the surface to determine the target's location. Determination of heading in this implementation is restricted to compass only due to the restriction of a

TABLE 4.1 : OBJECTS AND OPERATIONS ASSOCIATED WITH SANS

	OBJECT	OPERATION(s)
DR Navigation	Pitch Attitude	Current Value
	Depth	Current Value
	Heading	Current Value
	Delta_Lat_Long	DELTA_UPDATE
GPS Navigation	NAV_DATA_TYPE	PROCESS_GPS

single interrupt input channel in the Sapphire Converter. This restricts the ability to support a second driver to track rotation of the directional gyro.

The primary object associated with GPS is the class object NAV_DATA_TYPE. This object is generated by an operation PROCESS_GPS adapted from Dr. Kwak's functional implementation of a serial line interface in SNDL (see section II.D) consisting of latitude, longitude, position dilution of precision (PDOP) and time. Each position format message is processed to determine the position and PDOP to construct an instance of NAV_DATA_TYPE. The lowest level of PROCESS_GPS is the message interpreter MOTOROLA which replaces SNDL interpreters for other receivers. These receivers produce incompatible GPS message formats. Table 4.1 summarizes the objects essential to the operation of SANS at the highest level along with their associated operations.

b. Identify the Operations

The only operation associated with the basic elements of DR navigation is CURRENT_VALUE of that object. For heading and depth, this is accomplished through module A_TO_D. This module outputs a 12-bit digital value which is proportional in the range 0 through 4095 to the analog output of the respective instrument. These values are translated into appropriate units and written to non-volatile memory by CURRENT_HEADING and CURRENT_DEPTH respectively. Pitch attitude is tracked by an interrupt-driven service routine which may be queried for CURRENT_PITCH at any time by the main procedure.

c. Establish the Visibility

MONITOR_AUV interfaces directly with the AUV through a serial line interface to determine when an object of interest has been located by the AUV. It has no visibility of PROCESS_GPS or DR_NAVIGATION but simply relinquishes control to them when an update is requested. CURRENT_PITCH is invisible to all other operations as is PROCESS_GPS. Heading and depth analog to digital conversions both use A_TO_D but have independent routines for conversion to appropriate units.

d. Establish the Interface

The interface of objects is described fully by the specification of each package contained in the source code for SANS in Appendix B. Figure 4.2 illustrates the order of module dependency.

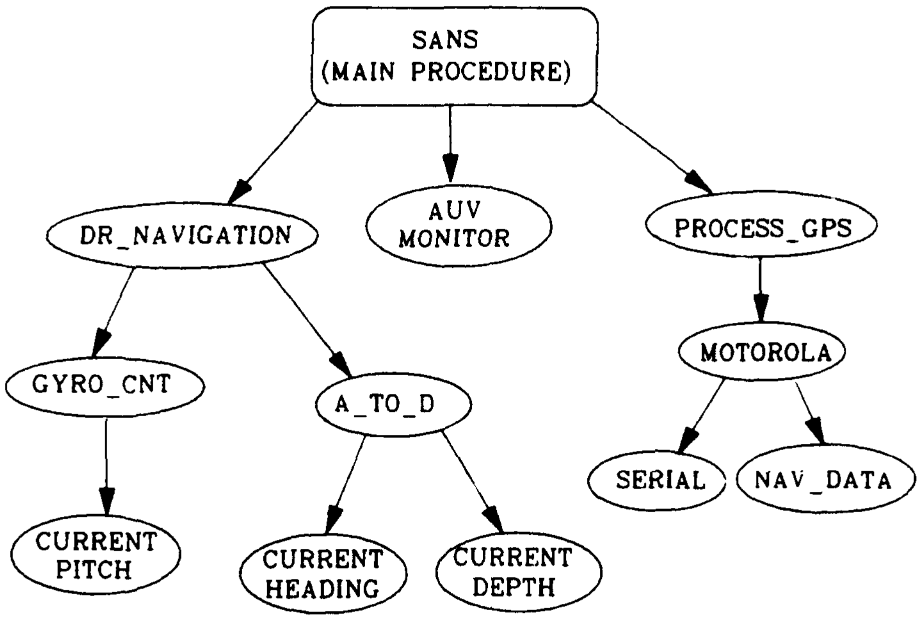


Figure 4.2 : Module Dependency Diagram

e. Implement Each Object (Analog Conversions)

The Sapphire A to D converter accepts control parameters through control registers 1 and 2 which are mapped to the I/O address space of the host CPU. The I/O address space map is shown in Table 4.2. Control registers 1 and 2 are accessed by offsetting the base address (factory preset at 300 Hexadecimal) by 2 and 3 respectively. Meridian Ada package Port provides procedure OUT_BYTE which writes a byte to the specified output port and function IN_BYTE which reads a byte from the specified input port.

TABLE 4.2 : SAPPHIRE I/O ADDRESS SPACE MAP

Offset From Base Address	Write	Read
0	8-bit A to D Trigger	4 LSB's from A to D
1	12-bit A to D Trigger	8 MSB's from A to D
2	Control Register 1	Status Register
3	Control Register 2	No Function
4	Counter 0 Register	Counter 0 Register
5	Counter 1 Register	Counter 1 Register
6	Counter 2 Register	Counter 2 Register
7	Counter Control Register	No Function
8	Port A Output	Port A Input
9	Port B Output	Port B Input
10	Port C Output	Port C Input
11	DIO Control Register	No Function
12-15	No Function	No Function

The sequence of operations in the A to D conversion is [Ref. 10]:

(1) Control register two is used to program the voltage range to be used (0 - 10 volts). The three least significant bits (LSB) in control register one select the multiplexed channel (0 to 7) matching the desired analog input port being utilized.

(2) After specification of the channel and voltage level to be used for the conversion, a switching time (minimum 8.5 micro-second delay) is programmed to

allow the multiplexed channel to stabilize at the analog input level. The conversion is then triggered by writing at the base address with an offset + 1 (301 hexadecimal). The written data can be any value but an ASCII character 1 is used for this specific application.

(3) After the conversion has been triggered, the conversion status register's most significant bit (MSB) indicates status of the conversion (1 indicates complete). This register is located at I/O address 302 hexadecimal (base address + 2). The register's contents are read in a loop using package Port's function IN_BYTE until the MSB indicates conversion complete.

(4) The conversion result is read from the two analog output registers using IN_BYTE. The 4 MSB's from the base address provide the conversion result's 4 LSB's (called the LSBS). The byte from base address + one provides the 8 MSB's (called the MSBS). The digital output value is determined by applying equation 4.1.

$$Value = MSBS * 16 + \frac{LSBS}{16} \qquad (4.1)$$

The division by 16 in the LSBS simply removes the 4 LSB's from the register contents. The multiplication by 16 in MSBS shifts this byte 4 bits to the left.

f. Implement Each Object (Digital Phase Counting)

Pitch angle is one object used in the computation of horizontal distance travelled from the datum as shown in Table 4.1. Digital phase counting of gyro phase and quadrature signal outputs is accomplished by initialization of the interrupt service

routine. This requires that the interrupt vector table be modified so that the interrupt level used to interface the gyro input may be handled by the service routine. The service routine implements the state transition algorithm developed below to track valid state transitions. The service routine and algorithm are procedure oriented in nature. Therefore, a functional implementation for this object is appropriate.

Development of the device driver for the gyro was accomplished in three major phases. Initial development concentrated on determining the characteristics of the phase and quadrature output of the gyro through development of a timed polling system. The data collected by the polling system was used to develop and test a state transition model and algorithm to filter out noise and track gyro rotation through valid state transitions. Finally, after development of the state transition algorithm, an assembly language interrupt routine was developed with a service routine adapted from the state transition model.

(1) Polled Sampling System. A polled system sampling both outputs of a particular gimbals on a 1.0 millisecond interval was developed using the Sapphire 82C54 programmable interval timer. The phase and quadrature signals are designated lines A and B by Gyration's documentation. [Ref. 13] As illustrated by Figure 3.3, clockwise rotation of the gyro is indicated by a phase shift to the right (the level of signal A will still be low after signal B has risen). For each pulse, 0.4 degrees of rotation (0.8 degrees in low-resolution gyros) has occurred and the event counter is

incremented to reflect this. Counter-clockwise rotation is indicated by a phase shift to the left (signal A has risen before signal B) and the counter is decremented.

The phase and quadrature output of the gyro is interfaced through the Sapphire card in the Digital I/O (DIO) mode of operation. The inner gimbals output is used for pitch attitude information. Inner Gimbal A (IG-A) is interfaced through IP1 (pin 25 of the DB-37) and Inner Gimbal B (IG-B) is interfaced through IP2 (pin 26).

A 1.0 milli-second polling interval (1,000 Hz) represents a rate which significantly exceeds the gyro's highest rate of 10,000 rpm (167 Hz) at 5.0 volts. The 82C54 (programmable interval timer) in the Sapphire card is used to time the read operations. Mode 3 of the 82C54 provides a square wave through any of the 3 counters in the 82C54. An initial count N is written to the counter control register (base offset + 7) and the counter decrements the loaded value N until this value reaches zero. When this occurs, the counter produces one pulse and reloads the counter with initial count N. This sequence of operations repeated results in a square wave with a period of N clock cycles. Counter 2 is used for this application since it's clock input is connected to the host computer's I/O bus clock. Therefore, calibration of the timer is dependent on the host's I/O bus clock speed. The 1.0 milli-second interval on the host machine with a 8.0 mHz I/O clock is achieved by programming an

$$\frac{8{,}000\,CYCLES}{8{,}000{,}000\,\dfrac{CYCLES}{SECOND}} = \frac{1}{1{,}000}\,SECONDS \qquad (4.2)$$

initial count (N) of 8,000 into the counter control register. This produces a square wave with a period of 8000 clock cycles so that the gyro input is polled at a rate of 1,000 Hz at 8.0 mHz as shown in equation 4.2.

The current value of the digital inputs are read through the status register (Sapphire base offset + 2) with bits 5 and 6 representing quadrature input values at the time of the sampling. To extract the values of the 4 MSB's from the 8-bit input, divide by 16 to remove the 4 least significant bits (LSB's).

$$INPUT\_BITS5\text{-}8 = \frac{INPUT}{16} \tag{4.3}$$

Take the remainder of a division by 4 to extract bits 5 and 6.

$$INPUT\_BITS5\text{-}6 = REMAINDER\frac{INPUT\_BITS5\text{-}8}{4} \tag{4.4}$$

Bit 6 is the result of a division by 2 and bit 5 is the remainder of a division by 2.

$$INPUT\_BIT5 = REMAINDER\frac{INPUT\_BITS5\text{-}6}{2} \tag{4.5}$$

$$INPUT\_BIT6 = \frac{INPUT\_BITS5\text{-}6}{2} \tag{4.6}$$

The current phase and quadrature input is read once at the beginning of each 1 milli-second counting cycle. After the current phased quadrature inputs are recorded, the value in counter 2, which is accessed at I/O address 306 hexadecimal (base address + 6), is monitored until the counter cycle ends. Specifically, to read the count, the counter must first be "latched". This records the counter value for the read operation while allowing the counter to continue to decrement. To determine when a cycle has expired, the counter register value is monitored. When the value in the counter data register reaches zero, the counter control register writes back the initial count N (8,000). When this rise in the counter register is detected, the cycle has expired and 1.0 millisecond has elapsed. At this point, the status register is read to determine the current phase and quadrature signal levels and the sequence begins again.

The sequence of operations is the polled state transition counting is:

(a) Write the initial count (N = 8,000) to the counter control register to establish a 1.0 millisecond interval on an 8.0 mHz I/O bus clock.

(b) Read the status register of the Sapphire board and extract bits 5 and 6 (equations 7 through 10) to determine the current phased quadrature value of IG-A and IG-B.

(c) Monitor the counter by latching counter 2 and reading the value. Repeat this process until the counter is incremented back to the initial count (8,000) then return to (a).

(2) State Transition Model Development. After the characteristics of the phase and quadrature output were determined through the timer based polling system, a state transition model was developed to track valid pulse pairs while filtering out "noise". The state represents the most recent value of the gimbals signals A and B where 0 is low (no pulse) and 1 is high. The state transition input similarly represents the new value of the gimbals signals and will always result in a transition to the corresponding state. The output is dependent on the path travelled in the transition, 1 for valid state transitions and 0 for invalid transitions.

If the states of the lines A and B are a and b, then the system state can be denoted ab. There are 4 possible states described by all combinations of a and b. Because the software senses only transitions of either a or b from 0 to 1, the state 0,0 is not visible to the software. Therefore, the software is designed around the three-state transition model in Figure 4.3. Figure 4.4 and Table 4.3 describe the tracking algorithm for the reduced state transition model.

The interrupt driven approach to digital phase counting results in two observable events in one period of the optical encoder's output. The beginning of a period is indicated by a transition to state 1 or state 3 since a pulse has occurred in either signal output. If the other output signal produces a pulse while the first signal is still high, a transition to state 2 will occur. In this case, the pitch count will be incremented or decremented as appropriate. Otherwise, the signal returns to state 1 or state 3 and the event is disregarded in the pitch count. Each unit of pitch count in the

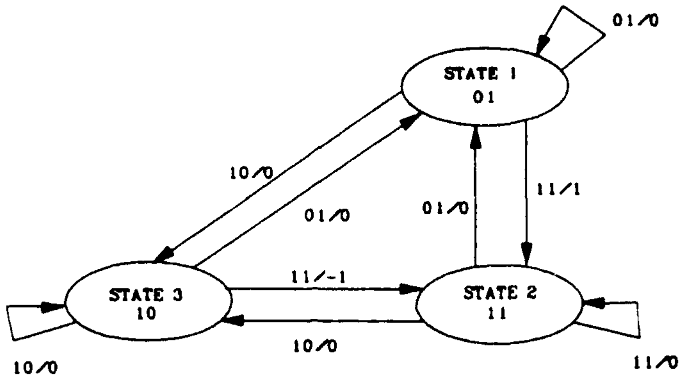


Figure 4.5 : Interrupt Driven Three-State Transition Model

```
procedure 3_state_tracking_algorithm is
   current_state, last_state : state(1..3)
   pitch_count : integer := 0

begin
  loop
     if current_state = 2 then
          if last_state = 1 then (case 3, clockwise rotation)
               increment pitch_count
          elsif last_state = 3 then (case 1, counter-clockwise rotation)
               decrement pitch_count
          end if
     else
        no trackable rotation has occurred (case 0, 2 or 4)
     end if
     last_state := current_state
     get(current_state)
  end loop
end 3_state_tracking_algorithm
```

Figure 4.6 : Three-State Transition Model Tracking Algorithm

three-state algorithm represents one period of the optical encoder (0.8 degrees of rotation in a low-resolution gyro). Verification of the algorithm is contained in Chapter V (Software Testing) and Appendix D.

(3) Interrupt-Driven Solution. The interrupt routine is modelled loosely after the serial line service routine developed by Dr. Kwak and used for GPS interface in SANS. LPT1 (level 7) is used as the interrupt number to interface interrupt inputs. The interrupt service routine implements the reduced state transition model which compares the current and previous state signals to track gyro rotation. [Ref. 12]

The interface of IG-A and IG-B (inner gimbals lines A and B respectively) is the same as for the polled system with an additional interrupt signal input. IG-A and IG-B are sampled immediately after each interrupt. Sapphire provides one interrupt input channel and the interrupt service routine requires rising edge trigger on either signal (IG-A or IG-B). An integrated circuit was developed by Dr. Kwak [Ref. 14] to generate distinct leading edges for interrupt triggers.

Each gimbals signal is first processed by a dual one-shot (74LS221) integrated circuit (IC). This IC generates a 10.0 microsecond pulse on the leading edge of the input signal pulse. The two signals are combined through a quad OR IC (74LS32) and input through INT.IN (pin 24 of the DB-37). The 10.0 microsecond pulse triggers the interrupt service routine. The service routine reads the value of each gimbals signals. These values are compared with the previous signal values as

illustrated in Figure 4.4 to determine what rotation, if any has occurred. A review of the major operations in the interrupt routine are contained in Appendix C.

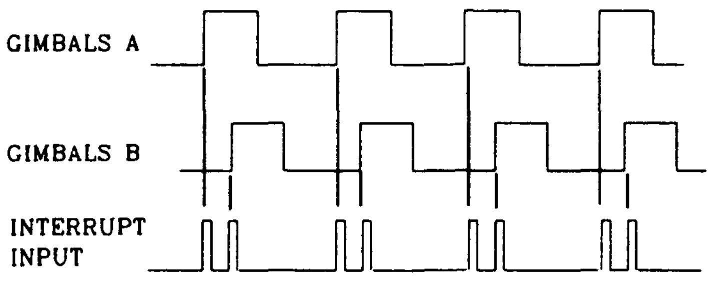


Figure 4.5 : Phase and Quadrature Interrupt Input Signal Generation

The driver developed performs very well as expected with the gyro in motion and at rest. The solution appears practical for the short duration, highly dynamic operations intended for SANS as verified in chapter VI.

3. GPS Update

The Suitcase Navigation Data Logger (SNDL) designed by Dr. Se-Hung Kwak establishes the framework for an RS-232 serial port interface to a GPS receiver. This design was kept largely intact for the GPS interface. The lowest level packages involving the GPS message translation were completely redesigned due to differences in receiver message format. The basic design philosophy for GPS processing in SNDL remains unchanged in SANS.

The AUV may request a GPS update during transit phase for a navigation check or during the mission execution when an object of interest has been located. As with

the gyro driver, the GPS serial line communication is established at each GPS update request and is terminated at completion of that update.

a. Serial Port Communication

When SANS detects an update request from the AUV, the GPS is powered up and INIT_GPS establishes the serial connection with the GPS receiver. The parameters for the connection are specified in the file SETUP.DAT. The initialization routine establishes a buffer where the interrupt service routine writes the bytes received through the connection.

Motorola proprietary binary message format was chosen for data transmission from the receiver. This is the only format which allows transmission of both position information for real-time navigation and satellite range information for post-processing.

PROCESS_GPS permits the main procedure to test for a GPS message by checking the buffer established during initialization through procedure READ_CHAR. The result of this procedure includes a status value which indicates whether the buffer has any new data (messages). When data is in the buffer, all data is processed before exiting to wait for another message. READ_CHAR is an assembly language program linked with the SANS program's Ada object code through pragma interface. Each character read through the serial interface is written to an output file for post-processing before any real-time processing.

b. GPS Message Translation

There are two message formats transmitted by the Motorola receiver. Position format messages are 68 bytes in length and are distinguished by the 4 byte header @@Ba. The 122 byte satellite range response message is recognized by the 4 byte header @@Bg.

If the first 4 bytes of the message match the position format message, the message is processed to produce an instance of NAV_DATA_TYPE. This is a composite type (record) consisting of latitude and longitude in degrees, minutes and seconds, PDOP and time as described in Section IV.B.2.a. Package Nav provides a procedure for translating milli-arcseconds in long-integer into degrees and fractions of degrees.

Bytes 16 through 19 encode a 32 bit two's complement long-integer. This value represents the latitude in milli-arcseconds in the range +/- 324,000,000 (+/- 90 degrees). Bytes 20 through 23 similarly represent the longitude in the range +/- 648,000,000 (+/- 180 degrees).

To convert the 32-bit two's complement number (4 bytes) to long-integer format, the parity must be determined by comparing the first byte with 127. If the first byte is greater than or equal to 128 (40 hex), then the first bit is one and the value therefore represents a negative number in two's complement form. In this case, all 4 bytes must be converted to a negative equivalent by subtracting 255 from each. A boolean flag is set to trigger the subtraction operation on each succeeding byte.

Each byte is multiplied by the appropriate factor as shown in equation 4.7 to determine the long-integer equivalent.

$$LONG\text{-}INTEGER = BYTE1 * 2^{24} + BYTE2 * 2^{16} + BYTE3 * 2^{8} BYTE4 \tag{4.7}$$

The value of PDOP, taken from bytes 36 and 37, provides the relative quality of the position to aid in integration into the update solution. Byte 66 contains a checksum which is the exclusive OR (XOR) of bytes 3 through 65. This value is compared with a checksum produced by the Motorola processor. If the two values do not match, the position is disregarded. Each binary format message is terminated with a carriage return and line feed (ASCII characters 13 and 10). By monitoring for these bytes, the end of the message is recognized so that processing can begin on subsequent messages.

c. GPS Fixing

There are two types of GPS fixes which must be considered in the design of SANS. During transit phase, real-time processing of GPS messages is required to maintain navigation accuracy of 100 meters. The ultimate goal of SANS is to provide GPS information for post-processing and combination with DR submerged navigation ascent vector to allow the location of submerged objects with an accuracy of 10 meters.

The message header distinguishes the message format as described earlier so that only position format messages are processed real-time by SANS for navigation updates to produce instances of NAV_DATA_TYPE. During transit mode, several instances of NAV_DATA_TYPE will be produced while the AUV is surfaced. When the accuracy of the fix is considered adequate by the main procedure, GPS processing will be terminated until the next update request from the AUV. This accuracy assessment will eventually be based on PDOP and relation to previous positions through a Kalman filtering technique.

V. SOFTWARE TESTING

A. INTRODUCTION

In order to provide confidence in the software architecture developed for SANS, a systematic and thorough if not exhaustive testing process is essential. Booch [Ref. 11] states two goals in the design of software tests:

Primary goal : Preventing bugs from entering code by making potential mistakes and misunderstandings visible and by identifying incomplete requirements/design.

Secondary goal : clearly identifying if there is a bug by causing that bug to produce a result that conflicts with a specified or expected result.

SANS is a multi-year project which will progressively incorporate new technology components to achieve improved accuracy. The solution described in this interim SANS design considers primarily the tactical mission execution of underwater mapping. The architecture will expand in the next generation to include the strategic considerations involved in transit to and from the target area and optimization of the mapping phase.

The interim SANS originally utilized Ada tasking to permit concurrent operation of DR Navigation and GPS processing. This concurrency was eventually eliminated since the calculation of the ascent vector and GPS processing occurs consecutively vice concurrently. The resulting structure is much less complicated at the highest level.

While the higher level architecture will probably undergo further radical changes to meet expanding requirements, modules developed in the interim system will be reused to provide a foundation for implementation of the next generation SANS. Due to the nature of development objectives and the elimination of concurrency, the testing approach here concentrates much more heavily on verifying the correct operation of individual modules than of the overall system. A functional (black box) testing approach is consistent with the objectives of ensuring modular correctness.

B. CLASSIFYING REQUIREMENTS

The software system requirements are developed from Section IV.B.1 (Software Architecture : Determining Requirements). Requirements can be classified according to the approach used to verify compliance. The four classification levels are non-testable, inspection (of source code), dynamic analysis (of variable usage, comments, etc.) and execution (of compiled code with selected test data). [Ref. 11]

1. AUV monitoring shall be performed continuously during the mapping phase so that when an object of interest has been located by the AUV, sufficient data may be recorded to determine the geographic position of the object through post-processing. This requirement can be verified by inspection and is satisfied through a loop in the main procedure which "idles" all processes until a position update request is received from the AUV. This loop continuously monitors an RS-232 serial connection with the AUV for an update request message. After the update is complete, the loop is then re-entered until another request is received.

2. DR navigation and GPS processing shall remain inactive with GPS unpowered (except for a low voltage applied to preserve volatile memory) until the AUV requests a position fix. Satisfaction of this requirement with respect to software is verified by inspection of the source code which indicates that DR and GPS processing only occur following an AUV update request. The hardware design must also ensure that power is physically removed from these devices between requests.

3. When an update is requested, AUV_MONITOR shall initiate DR navigation and GPS processing. Inspection verifies that upon receipt of an AUV update request, the AUV monitoring loop is exited so that DR and GPS processing can commence. DR navigation data is collected until the AUV is surfaced where GPS satellite range data is recorded.

4. The GPS shall be initialized with the receiver in the Motorola proprietary binary format with transmissions at a one second interval after the AUV requests a position fix. This requirement can be verified by inspection of source code and through execution of the initialization routine which initializes the receiver in the configuration which existed when power was last removed.

5. The receiver shall be programmed to transmit position/status format messages during transit phase and satellite range format messages during mapping phase when initialized. This requirement has not yet been satisfied but can be verified by inspection. Compliance will occur at implementation of transit phase during future research.

6. The host computer shall be configured to receive RS-232 serial communications with connection parameters specified from an input data file at run-time. This requirement is verified through inspection and execution of the serial communication procedure.

7. Upon mission completion, the host machine configuration shall be restored to the state which existed prior to SANS execution. Inspection shows that the original interrupt masks and vectors are stored at initialization in the data section of the assembly language code for serial connections and gyro driver. These values are restored by CLOSE_SERIAL upon program termination. Prior to program execution, the DOS (disk operating system) program debug.exe is used to verify the current interrupt vector for affected hardware interrupt levels. After termination, DOS debug verifies that the vector table has been restored to its original condition.

8. All information received through the serial connection by the host machine shall be written to non-volatile memory prior to real-time processing. Inspection of package body Motorola shows that this requirement is satisfied since each execution of READ_CHAR is followed by a write to the output file prior to any other processing.

9. Position format messages shall be processed to determine the current position and PDOP for navigation updates and to determine the checksum to validate the particular message. Execution test cases are developed in Appendix F to ensure that position and PDOP are correctly interpreted.

10. DR navigation outputs shall include pitch attitude and heading in degrees and depth in meters. By inspection of the specification of DELTA_UPDATE, the inputs for DR navigation are confirmed. These values are recorded in non-volatile memory along with mission time.

11. DR processing shall record output values with a precision commensurate with component accuracy and sufficient to satisfy system accuracy requirements specified in section I.D.d. Chapter VI examines component and system accurracy to include an assessment of overall system performance.

12. Gyro position shall be monitored continuously by a process transparent to SANS operation. Inspection of the assembly language device driver for tracking gyro rotation shows that interrupts generated from the phased quadrature input are used to trigger an interrupt service routine to count valid pulses. The machine state is saved at each interrupt and restored after the interrupt is handled. Execution of benchmark processes to examine CPU processing limitations for handling interrupts from the gyro driver are detailed in Section V.D.

13. The process shall take a pair of standard phased quadrature signals as inputs. This requirement is verified through inspection of the assembly language device driver for the gyro as in requirement 12. The signals are interfaced through pins 25 and 26 of the Sapphire DB-37 connector.

14. Valid pulse pairs [Fig. 4.3] shall be counted on a continuous basis with each unit representing +/- 0.4 degrees of rotation depending on the direction of rotation.

This requirement is satisfied through analysis in section V.D.1.a of the 3-state tracking algorithm [Fig. 4.4] contained in section V.E.2.a. Execution test cases contained in Appendix E ensure that the algorithm is correctly implemented.

15. Invalid pulses from either signal (non-overlapping) are considered "noise" [Fig. 4.4] and shall be disregarded in the pulse count. As in requirement 14, analysis verifies the correctness of the algorithm with execution test cases in Appendix E validating the implementation.

16. The cumulative value of the pulse count shall be continuously available for interrogation by the main procedure for DR navigation. Verification of this requirement is accomplished through inspection of the assembly language gyro driver and inspection of the interface between the gyro driver and the main procedure.

17. Sapphire A to D conversions shall be initialized from a data file containing the multiplexed channels to be used for heading and depth inputs. Inspection of the main procedure and package ANALOG shows that the A to D conversion channels are selected from the input data file A_TO_D.dat.

18. Analog to digital conversion of analog heading and depth output shall provide 12 bit accuracy (one part in 4096). Execution of a range of analog inputs from a DC power supply were used to correlate voltage levels with digitized outputs. Inspection of the implementation in package ANALOG shows that 12 bit accuracy is provided.

C. IDENTIFY IMPORTANT CASES AND SELECT TEST DATA

1. Requirements 13 through 16 involve the accurate tracking of gyro rotation and is evaluated in two phases. The state transition tracking algorithm is first verified by informal proof in section V.D.1.a below. Test cases read from an input file are then executed to validate the implementation of the algorithm. Inputs with expected and observed results are contained in Appendix D.

a. There are 4 cases to examine in the 3-state transition tracking algorithm as shown in Figure 4.4. Appendix D contains examples of each state transition case along with results from test executions.

In cases 1 and 3, valid rotation has been indicated by a state transition from state 1 (clockwise rotation) or from state 3 (counter-clockwise rotation) to state 2. This is illustrated in Figures D.1 and D.2 respectively. The results of the test execution demonstrate that the pitch_count is correctly computed in each case. Figures D.3 and D.4 demonstrate that the algorithm correctly tracks a reversal in rotation as illustrated by a reversal in the order of rising edges in the gimbals output signals.

In case 2 transitions, the machine is returning to state 1 or 3 from state 2. This is usually an indication that a valid rotation has occurred in the previous sampling and a new pulse has been detected from one signal input. Figures D.1 through D.4 demonstrate that case 2 transitions are correctly discounted by the implementation.

In case 0, no state transition has occurred since the current state is the same as the last state. This is typically the case with noise in a single channel. Figures D.5, D.6 and D.7 demonstrate that no rotation is occurring in case 0 state transitions and also that this case is correctly discounted by the tracking algorithm's implementation.

Case 4 transitions are invalid state transitions between state 1 and state 3. This transition is caused by a pulse from one gimbals then the other where the pulses do not overlap. The likely cause of a case 4 transition is noise in both channels as illustrated in Figure D.8. Figure D.8 also demonstrates that this case is correctly discounted by the tracking algorithm.

2. Requirement 8 calls for the correct interpretation of the latitude, longitude and PDOP from position format messages. This requirement is evaluated through a test implementation of package MOTOROLA which reads a binary input file vice actual receiver messages. In this manner, all paths can be tested and a full range of edge cases can be executed.

The range of latitudes is +/- 324,000,000 degrees in milliseconds. For longitude, the range is +/- 648,000,000 degrees in milliseconds. Test cases include positive and negative two's complement values to exercise hemisphere interpretation and various message formats. Appendix E summarizes test cases with expected and observed results.

D. REAL-TIME PROCESSING VERIFICATION

The elimination of concurrent processing from the main procedure in the interim SANS has greatly simplified the timing requirements at the highest level. In the lowest level assembly language procedures employing interrupts, a statistical analysis is conducted here to evaluate the adequacy of the software development machine to handle the expected interrupts and processing requirements. Growth capability is examined for handling multiple gyros.

The software development machine was benchmarked for 10,000 sine operations to evaluate the effect of the interrupt service routine on central processor unit (CPU) availability. Execution of the sine operations with no other significant load required 306.795 seconds with a standard deviation of 0.085 seconds. After the sine operations were benchmarked, the driver for the gyro was loaded. Both the pitch and roll axis of the vertical gyro were interfaced through the single interrupt input channel to simulate multiple interrupt inputs.

The time to execute the same 10,000 sine operations with both gyro axis highly excited increased by 6.035 seconds with a standard deviation of 0.41 seconds. This is an increase of only 1.97 percent in execution time. The only other time critical processing requirement in SANS is the RS-232 (9600 baud) serial line connection which imposes a similar CPU load. The software development machine should handle such loads with no problems. These conditions should be reassessed when the software and hardware are moved to the ESP-8680 target machine.

VI. HARDWARE CHARACTERISTICS

All accuracy assessments made in earlier research are based on manufacturer's technical specifications. In order to validate these findings, field and laboratory test data are presented here with statistical analysis of those results. This chapter concludes with an assessment of overall system performance based on these findings.

A. MOTOROLA GPS RECEIVER

A major limiting factor in SANS accuracy is the accuracy of the GPS receiver. Extensive GPS data was collected with the Motorola PVT6 receiver to assess the impact on system performance. There are two broad test categories conducted in this research involving GPS. These tests evaluate the receiver for accuracy in a dynamic environment and for acquisition time in a static environment. The major limitation on the test results presented here is the absence of differential processing which has been deferred to future research.

1. GPS DYNAMIC TESTS

GPS dynamic tests were conducted onboard a modified electric golf cart (called the test vehicle). This platform is capable of a top speed of approximately 5 meters/second and provides both 12 volt DC from a deep marine battery and 110 volt alternating current (AC) through a DC to AC converter.

In order to assess the accuracy of field data, the test vehicle is specially instrumented to permit accurate tracking through a surveyed test track. Data is logged

by a general serial I/O handler developed by Dr. James Clynch called GEORGE. This program provides a wide variety of functions including a printer port status (PPS) which logs changes in the condition of the instrumentation devices interfaced through the printer port. [Ref. 15]

Test vehicle instrumentation includes an infrared (IR) sensor which detects reflective IR strips on the surveyed test track. The position of the centroid of the IR strips is known to an accuracy of approximately 1 centimeter. The vehicle is also equipped with a trailing bicycle wheel with magnetic sensors for recording longitudinal distance travelled and a "button" which allows the operator to manually mark some event. Dynamic tests were conducted at Fritzsche Army Air Field (AAF) at Fort Ord, California. The test track was surveyed to an accuracy of approximately 1 centimeter (cm).

The objective in the dynamic GPS data collection is to assess the dynamic accuracy of the Motorola GPS receiver relative to expected SPS accuracy. Fourteen test runs over a one hour period were conducted to gather data. Figure 6.1 illustrates the positional accuracy of the test runs relative to the known course. This is a plot of unprocessed GPS without respect to time.

In order to assess the accuracy of the GPS data, a comparison was made with test track events recorded by GEORGE's operation PPS. This time history will be called "truth data" due to its known relative precision of 1 centimeter. The host computer was corrected to universal coordinated time (UTC) via modem connection to

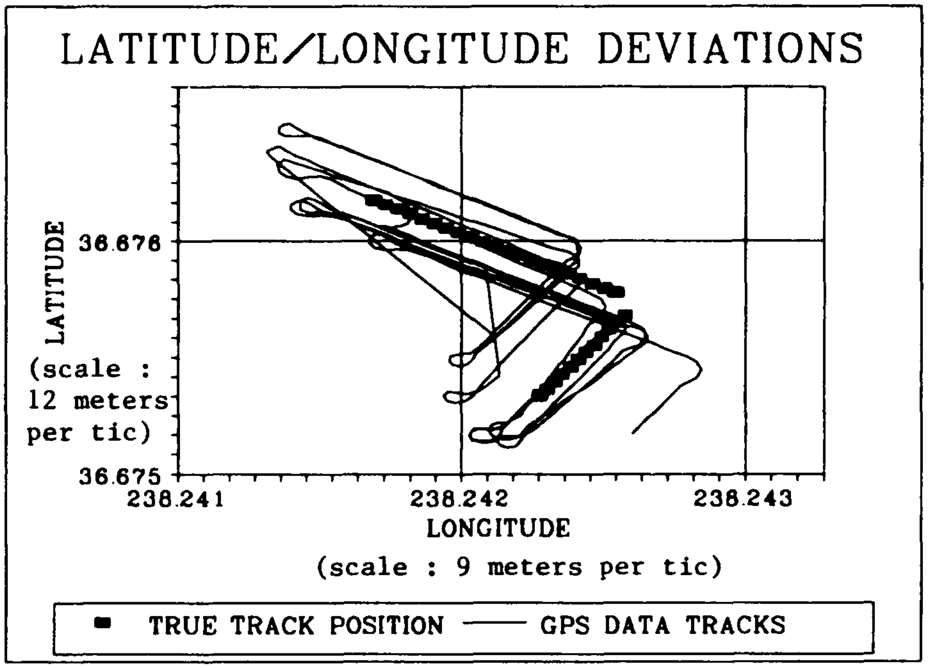


Figure 6.1 : Test Data Results Relative to Known Positions

permit correlation of GEORGE's time reference with GPS time (also recorded in UTC).

The GPS binary data was processed through package Motorola's test procedure to convert data of interest to character format for comparison with GEORGE's "truth data". GEORGE's PPS data was similarly processed to allow for comparison. The raw PPS file was first processed by package BIW (button, IR, Wheel) to identify each event according to its source. The BIW file was then manually processed with reference to the sequence of events recorded at the time of test. The purpose here is to identify the specific location of each IR marker in the "truth data" file for

comparison with GPS positions. The velocity was computed based on wheel revolutions with respect to elapsed time.

Finally, the processed GPS and truth data were correlated using a routine developed by Dr. Clynch called DIFDAT. This procedure compares a specified parameter, in this case time, in each file to interpolate between events occurring asynchronously in the two files. This permits an assessment of the accuracy of the GPS data relative to the known positions in the truth data even though the position times do not correlate exactly. [Ref. 16]

As can be seen in Figures 6.2 and 6.3 the standard deviation of error is 18.7 meters in latitude and 13.2 meters in longitude. This yields a positional error of 22.9 meters one rms which is well within the expected range for SPS accuracy. North and east velocity components were similarly processed from the raw GPS and truth data files. The error is 0.21 meters/second for north velocity components and 0.17 meters/second for east velocity components. This yields a rms velocity error of 0.27 meters/second.

2. GPS STATIC TESTS

Static tests were conducted using the Motorola receiver to assess acquisition time and first fix accuracy. These tests utilize an antenna mounted on the roof of Spanagel Hall at a surveyed location. This site provides a horizon relatively free from obstructions.

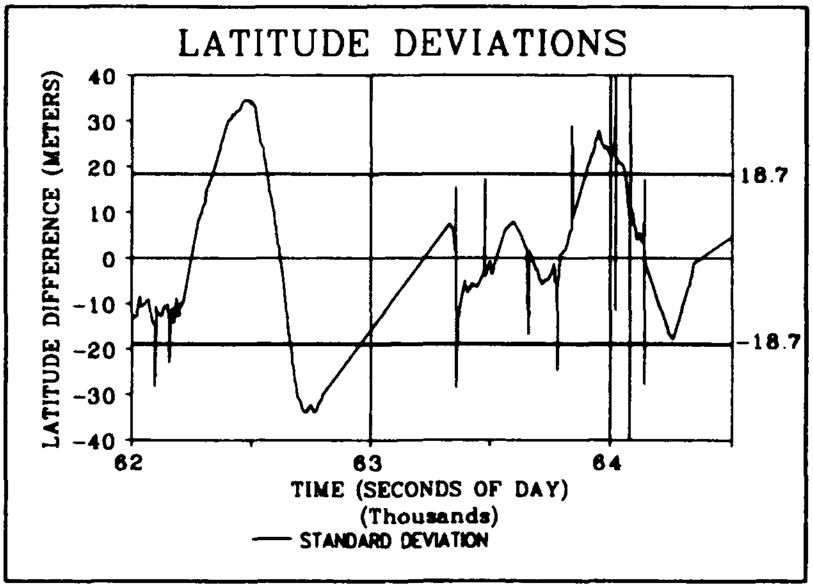


Figure 6.2 : GPS Dynamic Data Latitude Deviations

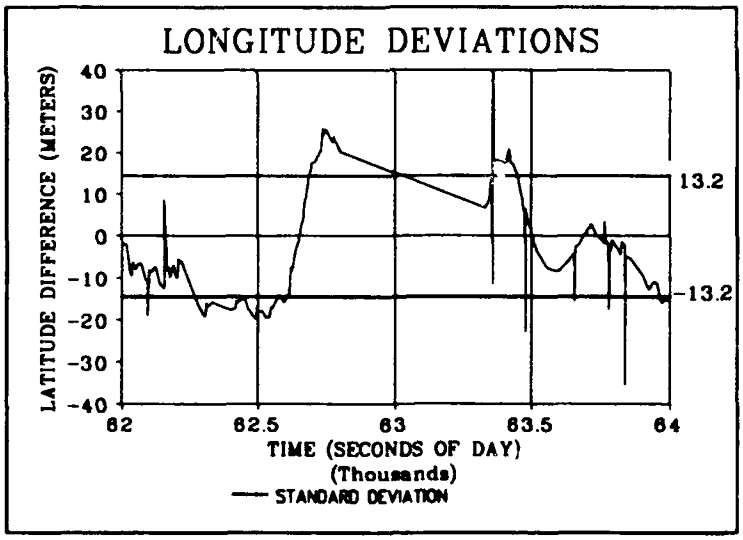


Figure 6.3 : GPS Dynamic Data Longitude Deviations

GEORGE's operation PPS was used to program a relay box which interrupts the 12 volt main power supply to the GPS receiver for a specified duration and interval. However, the 5 volt regulated keep alive power is supplied continuously to preserve the most recent coordinates and constellation in SRAM volatile memory. GPS data is logged using the manufacturer's software to ease processing requirements due to compatibilty with commercial spreadsheets.

Appendix F illustrates the results of these tests as a function of the amount of time which the receiver was unpowered. These times range from 90 seconds through 6 hours. The results are summarized in Table 6.1. Although time to first fix (TTTF) is relatively stable through the longest test (6 hours), the accuracy of the first fix degrades considerably beyond 30 minutes off.

TABLE 6.1 : GPS STATIC TEST RESULTS SUMMARY

RECEIVER TIME UNPOWERED	ACQUISITION TIME (rms) SECONDS	FIRST FIX ERROR (rms) METERS	NUMBER of SAMPLES
90 SECONDS OFF	29.1	46.4	45
10 MINUTES OFF	29.6	51.2	30
30 MINUTES OFF	29.2	48.6	39
1 HOUR OFF	28.7	60.9	33
3 HOURS OFF	42.0	75.2	23
6 HOURS OFF	49.4	25.0	15

Analysis of the raw GPS data indicates that the predominant factor in this accuracy degradation is occasional poor geometry in the first set of satellites acquired by the receiver. Inspection of tracked satellites indicates that the geometry improves quickly after tracking the first satellite due to rapid acquisition of additional satellites. However, this process continues to slow as time unpowered increases due to increasing frequency of changes in the currently visible satellites.

The static test results indicate that the TTTF is relatively stable below 30 seconds until the time off exceeds one hour. The first fix accuracy deteriorates steadily as a function of time off with the exception of 6 hours off where the TTTF is excessive. The maximum time between GPS updates should not normally exceed approximately 30 minutes in order to improve first fix accuracy. At this update frequency, antenna exposure can be limited to approximately 30 seconds maximum. Longer intervals between updates are possible but will necessitate longer antenna exposures to obtain a position update of reasonable accuracy.

These test results confirm the manufacturer's specifications in Appendix A and indicate that the objectives for the SANS transit phase are easily attainable. Further testing using differential processing techniques is required to validate feasibility of the accuracy requirements for the mapping phase.

B. GYRATION GYROENGINE

Testing of the gyroscopes was conducted at the AAF test track. The testing was conducted in cooperation with the manufacturer of the gyro and consisted of 10 test

circuits on the instrumented test track. Only the directional gyro was evaluated due to the two-dimensional orientation of the test track. Gyro data was recorded utilizing the manufacturer's software. GEORGE was again used to record "truth data" from the test vehicle. Processing of the "truth data" was conducted in the same fashion as for the dynamic GPS experiments. This data was then compared manually with the gyro data to determine the cumulative error for each individual run.

The test gyro was powered by 12.0 volts during 8 of 10 test runs. Single test run were conducted at 5.0 volts and 15.0 volts. During steady state operations at 12.0 volts, the mean drift rate was 2.3 degrees per minute (dpm) with a standard deviation of 2.2 dpm yielding 3.2 dpm rms. This is slightly higher than the product specification for Scorsby drift rates of 1.5 dpm typical and 3.0 dpm maximum [Ref. 13].

During the 9.2 meter radius turn at the corner of the test track with an average speed of 3.0 meters per second, the average acceleration was computed to be 0.16 G's or 1.6 meters per second per second. The observed drift rate was 25.4 dpm with a standard deviation of 15.6 dpm yielding 29.8 dpm rms. The expected drift rate at the advertised specification of 3.0 degrees per second per G for the computed 0.16 G is 30 dpm. These observed and expected results here are consistent and demonstrate that acceleration is the dominant factor in gyro drift rate.

The result of a single test run at 5.0 volts DC is particularly noteworthy. Although steady state drift rate was consistent with other tests, the drift rate in turns was 120

dpm. The high drift rate at this low motor speed is unacceptable for the intended applications in SANS. Since the SANS driver for the gyro was developed for execution at 5.0 volts, future research should concentrate on expanding the envelope of the SANS gyro driver with higher voltages and the resultant higher motor speeds. Further testing is also necessary to validate the performance of the pitch axis of the vertical gyro.

C. DEPTH TRANSDUCER

The Omega Inc., depth transducer was evaluated on a 50 PSI Chandler Engineering dead weight tester. This test bench utilizes a hydraulic cylinder to pressurize the diaphragm of the depth transducer. Pressure is generated in increments of 5 PSI by adding calibrated weights to pressurize the cylinder. The depth transducer's output was measured by a digital voltmeter to 3 decimal points accuracy (milli-volts).

Figure 6.4 illustrates the results of the pressurization schedule. Due to the minimum pressurization of 50 PSI, no measurements are possible between ambient and 50 PSI. The maximum deviation from a linear function in the digital output is observed at the 50 PSI mid-point. The deviation at this point is 16 milli-volts (3.516 volt average over 4 runs with 3.499 volts expected). This deviation correlates to 0.46 percent, which is within the 0.5 percent advertised linearity accuracy. [Ref. 17] For SANS, this translates into 0.335 meter accuracy for full depth excursions.

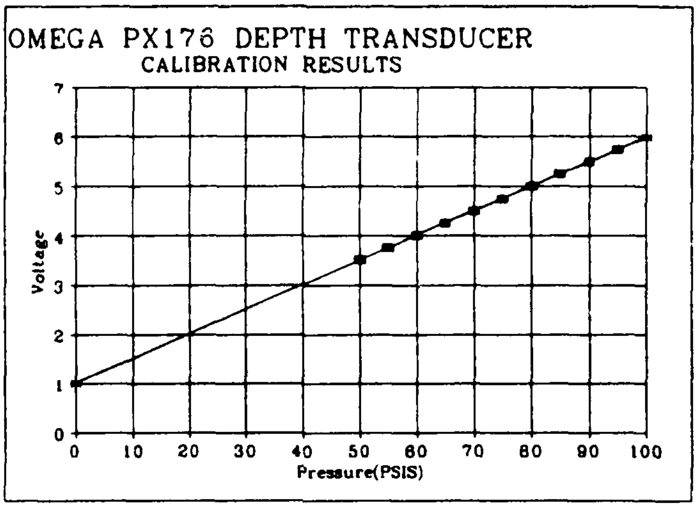


Figure 6.4 : Depth Transducer Calibration Results

D. SYSTEM PERFORMANCE ASSESSMENT

1. GPS Performance

Although not all components of the system have yet been evaluated in their operational mode, some assumptions can be made to assess the overall performance of the interim SANS. The results of the GPS static testing certainly indicate that the target acquisition times and fix accuracy are attainable for update intervals of less than 30 minutes. These results also indicate that differential post-processing should permit GPS positions to be determined to an accuracy of 2 to 4 meters.

2. DR Navigation Performance

The accuracy of the computation of the horizontal distance travelled is affected by the accuracy of th· depth transducer and gyro (or accelerometers) during the ascent. The maximum effect of the pitch attitude inaccuracy would occur wh'le SANS is level

(pitch attitude = 0 degrees) since the value of tangent(theta) in the denominator of (Equation 3.2) approaches 0.0 at zero degrees. This would result in an error of infinite magnitude for even small errors in pitch attitude measurement. Similarly, heading measurement inaccuracy affects system accuracy most when the pitch attitude is very shallow since this results in a large horizontal distance travelled and high gyro drift.

The differential approach used here to assess the overall accuracy of SANS was developed by Dr. J. R. Clynch. [Ref. 18] The analysis uses the manufacturer's technical specification in Appendix A except in the case of the gyro drift rate where the slightly higher test results are used. The drift rate for the pitch axis of the vertical gyro is assumed to be the same as that calculated for the directional gyro.

Recall from Figures 3.1 and 3.2 and the text that the climb angle (theta) is measured by the pitch axis of the vertical gyro. Alpha is the AUV heading which will be measured by compass and directional gyro in future development. Let h represent the change in depth measured by the depth transducer and let s_H represent the horizontal distance travelled by the AUV from a submerged object of interest during its ascent to the surface where the GPS position will be determined. Then as in equation 3.2:

$$s_H = \frac{h}{\tan\theta} \qquad \textbf{(6.1)}$$

By taking the first derivative:

$$ds_H = dh\frac{\cos(\theta)}{\sin(\theta)} - \frac{h}{\sin^2(\theta)}d\theta \tag{6.2}$$

Because the depth and pitch angle errors are uncorrelated, the sum of the magnitude of the two terms in equation 6.2 can be used to estimate the size of the error in slant range. [Ref. 18] Equation 6.3 yields the position error due to precession in the heading gyro.

$$AOU = s_H * d\alpha * ds_H \tag{6.3}$$

Where AOU is the Area of Uncertainty.

The AOU will be the slant range error times the heading error. The radius of a circle covering the same area will be used as an estimate of the positional error. [Ref. 18] Figure 6.5 illustrates the computation of the positional error.

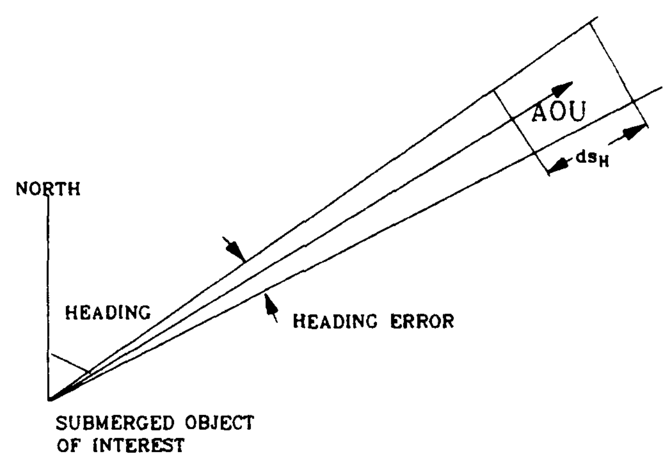


Figure 6.5 : Computation of Inertial Measurement AOU

Table 6.2 tabulates the results of error approximations for varying climb angles. A transit from the depth transducer's maximum depth is assumed with a maximum gyro drift rate of 3.2 dpm as calculated above (assuming no significant accelerations). The GPS error is assumed to be 4.0 meters rms.

TABLE 6.2 : SANS SYSTEM ACCURACY FROM 70 METERS

CLIMB ANGLE	20 degrees	25 degrees	30 Ddgrees
TIME TO CLIMB	39 seconds	32 seconds	27 seconds
GYRO PRECESSION	2.1 degrees 0.036 radians	1.7 degrees 0.03 radians	1.4 degrees 0.025 radians
AREA OF UNCERTAINTY	140.0 meters^2	49.0 meters^2	11.6 meters^2
POP-UP ERROR rms	11.8 meters	7.0 meters	3.4 meters
TOTAL ERROR rms (GPS + POP-UP)	12.5 meters	8.0 meters	5.3 meters

The objective for accuracy during mapping phase of 10 meters rms are achieved by SANS during transits from its maximum depth of 67 meters only when the climb angle is slightly greater than 30 degrees. NPS AUV2 is restricted to much shallower climb angles but also to depths of less than approximately 10 meters. This depth is more typical of the application expected for SANS. As indicated by Table 6.3, climb angles greater than 12 degrees should satisfy the accuracy objectives of SANS.

TABLE 6.3 SANS SYSTEM ACCURACY FROM 20 METERS

CLIMB ANGLE	10 degrees	12 degrees	15 degrees
TIME TO CLIMB	23 seconds	19 seconds	15 seconds
GYRO PRECESSION	1.2 degrees 0.02 radians	1.0 degrees 0.017 radians	0.8 degrees 0.014 radians
AREA OF UNCERTAINTY	101 meters^2	46.3 meters^2	4.4 meters^2
POP-UP ERROR rms	10.0 meters	6.8 meters	2.1 meters
TOTAL ERROR rms (GPS + POP-UP)	10.8 meters	7.9 meters	4.5 meters

VII. CONCLUSIONS

A. SOFTWARE ARCHITECTURE

A primarily object-oriented design approach is used to implement the design of SANS. The major operations performed by SANS are

- Monitoring the AUV for a position fix request
- Navigation data-logging for dead reckoning (DR) navigation (post-processed to determine ascent vector), and
- GPS data-logging for post-processed positional information.

The first and third of these are implemented primarily through a serial communication line routine designed by Dr. Se-Hung Kwak. The GPS processing routine was redesigned at the lowest level to accommodate the proprietary binary format used by Motorola. DR navigation implements a navigation data logger for hardware components of an inertial navigation system. This data includes the results of A to D conversions of compass and depth transducer outputs and D to D counting operations associated with phase and quadrature digital output signals from a miniature spin gyro. D to D operations are implemented in assembly language for higher efficiency due to their high frequency. A to D operations are implemented in Ada.

A major goal in the implementation of SANS has been modularity of code. As new technology is incorporated, this approach facilitates redesign of the entire system

with many modules being reused or replaced. Complexity management in a moderately large software system like SANS is also enhanced. [Ref. 11]

B. SOFTWARE TESTING

Requirements based testing is applied to SANS in order to verify the proper operation of key modules in the implementation. A primarily functional (black box) testing approach is used in keeping with the design goal of modularity to ensure that individual modules may be replaced or reused without side-effects on other modules. Some of the key findings are summarized below.

1. GPS MESSAGE TRANSLATION

a. All paths in package Motorola are reachable through the various test cases represented in Appendix E. All outputs are consistent with expected results. Validation of checksums are not yet implemented. This requirement will be essential for real-time processing of the position format messages in the implementation of the transit phase of the mission.

b. Values falling outside the ranges of +/- 648,000,000 for longitude and +/- 324,000,000 for latitude are possible if messages are incorrectly translated. Although testing for this situation is not specified as a requirement, correction is advisable and is accomplished by declaring legal ranges for input values.

2. GYRO DRIVER

a. Gyro rotation is correctly tracked through all possible state transitions as demonstrated in Appendix D. This result is consistent with many hours of direct

observation of pre-production gyros under dynamic operating conditions in which only low drift rate precession has been observed.

b. Interrupt handling for the gyro is easily accommodated in the software development machine. Real time processing requirements should be reassessed when the software and hardware are moved to the ESP-8680 target machine.

C. HARDWARE CHARACTERISTICS

All components meet or exceed all critical specifications except for the gyro where the drift rate of 3.2 degrees per minute is slightly higher than the maximum specified (3.0 degrees per minute). The error analysis conducted in section VI.D uses the more conservative results for the gyro rather than manufacturer's technical specifications. Further research is necessary to validate the performance of the pitch axis of the vertical gyro, the compass accuracy and the accuracy of post-processed GPS.

The error analysis indicates that all objectives of the interim SANS are achievable according to research conducted to date. The only major restriction associated with the interim SANS system is the requirement that the AUV must be restricted to a climb angle of slightly greater than 30 degrees from the SANS' maximum depth of 70 meters. During climbs from a more typical depth of 20 meters, a climb angle of approximately 12 degrees will deliver acceptable accuracy. These restrictions are necessary in order to ensure acceptable positional accuracy which is degraded by the increasing degree of gyro precession during the long ascent paths associated with

shallow climb angles. These restrictions will limit a technology demonstration to more shallow depths or to AUV's capable of attaining large climb angles.

D. FUTURE RESEARCH

1. SOFTWARE DEVELOPMENT

All software testing so far has been confined to the software development machine. The next critical phase in the software development is to evaluate the performance in the ESP-8680 target machine. Additional hardware is necessary to support the operation of SANS during this process. The A to D converter selected should have two interrupt input channels so that the yaw axis directional gyro may be tracked to improve heading accuracy during high turn rates. The current implementation of SANS is restricted to the pitch axis of the vertical gyro only.

The primary focus of this research has been the conduct of the mapping phase of the SANS mission. This only requires that satellite range format messages be recorded in order to permit the AUV's location to be determined through post-processing. In order to execute the transit phase of the mission, the Motorola GPS receiver must be reinitialized during the mission to transmit position format messages. This will be accomplished through transmission of preformatted binary messages using package SERIAL's function WRITE_CHAR for writing characters through the serial communication.

2. GYROSCOPES

As stated above, further research is necessary to validate the performance characteristics of the pitch axis of the vertical gyro. This will most likely require laboratory experiments using a servo-driven tilt table. [Ref. 19]

All testing with the device driver for the gyro thus far has been accomplished using pre-production items on loan from Gyration, Inc. During initial operation of the production issue gyros with the software development device driver, some inconsistencies in tracking gyro rotation were observed. Compatibility of the driver with these production items must be assessed. Expansion of the driver's envelope must also be addressed to ensure proper operation at higher input voltages and the corresponding higher motor speeds. Clearly, the field tests indicate that even under moderate acceleration at the low motor speed associated with an input of 5.0 volts, the drift rate is unacceptable for the application intended here. Further research may be necessary to develop a solution incorporating the use of accelerometers to dampen the effect of high frequency pitch excursions.

Finally, an applicability study of a new small but high performance IMU package developed by Systron Donner Corporation is proposed for SANS. This unit is expected to meet the space and power requirements of SANS and easily improve the system accuracy. [Ref. 19]

LIST OF REFERENCES

1. McKeon, James B. *Incorporation of GPS into Small Underwater Navigation System*, Master's Thesis, Naval Postgraduate School, Monterey California, pp 4-6, March 1992.

2. Naval Postgraduate School Report NPSCS-92-001, *Technology Survey and Preliminary Design for a Small AUV Navigation System*, McGhee, R. B. et al., pp 1-3, 13-18, 28-29, March 1992.

3. Clynch, J. R., "Dynamic GPS Solutions", Naval Postgraduate School Interoffice Memorandum (to files), August 1992.

4. Kwak, Se-Hung, "Suitcase Navigation Data Logger (SNDL)", unpublished Ada and Assembly computer language source code, September 1991.

5. Hutchinson, B. L. et al., *A System Approach to Navigating and Piloting Small UUV's*, pp 135, Proceedings of the Symposium on AUV Technology, June 1990.

6. Brown, G. B., *An Introduction to Random Signals and Applied Kalman Filtering*, pp. 437-441, John Wiley and Sons, Inc., New York, New York, 1992.

7. Nagengast, S. *Correction of Inertial Measurements Using GPS Update for Underwater Navigation*, Master's Thesis, Naval Postgraduate School, Monterey California, pp 4-6, March 1992.

8. Miller, C., *Application of Extended Kalman Filter to a Model-Based Navigator for an AUV*, Master's Thesis, Naval Postgraduate School, Monterey California, December 1991.

9. McGhee, R. B., "Finite Approximation of Climb Angle Using an Accelerometer", unpublished notes presented in a private lecture, Naval Postgraduate School, March 1993.

10. *Sapphire User's Manual*, pp. 4-5, 15-42, Diamond Systems Corp., 1992.

11. Booch, G,. *Software Engineering with Ada*, pp 48-49, The Benjamin/Cummings Publishing Company, Inc., New York, New York, 1987.

12. *IBM Technical Reference*, pp. 1-33, International Business Machines Corp., Boca Raton, Florida, September 1985.

13. *Gyration Development Manual*, pp. 3-4, 4-3, 4-4, Gyration, Inc., Saratoga, California, November 1992.

14. Kwak, Se-Hung, "Interrupt Signal Interface", unpublished circuit design for phase and quadrature interrupt signal generation, pp. 1, March 1993.

15. Clynch, J. R., "George 4.3 Documentation", Naval Postgraduate School, Monterey, California,unpublished software input/output handler, July 1992.

16 Clynch, J. R., "DIFDAT.EXE", Naval Postgraduate School, Monterey, California, unpublished computer software program.

17. "PX176 Series Pressure Transducer Data Sheet", pp. 1-2, Omega Technologies Limited, (no date).

18. Clynch, J. R., "Area of Uncertainty Approximation in Inertial Navigation Solution", unpublished notes presented in private lecture, Naval Postgraduate Schooi, Monterey, California, May 1993.

19. Healey, A. J., "Shallow Water Mine Hunting with the NPS AUV-2", Naval Postgraduate School, Monterey, California, Proposal For Research, pp. 7-8, April 1993.

20. *IBM Disk Operating System Technical Reference*, pp. 5-27, 5-31, International Business Machines Corp., Boca Raton, Florida, September 1983.

APPENDIX A

TECHNICAL SPECIFICATIONS

A. SAPPHIRE CONVERTER

Analog Section:

A/D resolution:	12-bits (1/4096)
Linearity error:	1/2 LSB max
Differential linearity error:	12 bits (no missing codes)
Conversion time:	20μs max
Maximum conversion rate*:	15kHz with clock timing 30kHz in software burst mode
Input channels:	8, single-ended
Expansion capabality:	128 channels, differential
Input ranges:	±5V, ±10V, 0-10V
Input range selection:	Software
Input impedance:	2.5MΩ min
Channel-to-channel isolation:	68dB min
Input overvoltage protection:	±32V

Digital Section:

Digital I/O:	31 bits, TTL-compatible
Direction:	4 out, 3 in (J1) Software configurable (J2)

Output voltage levels:

J1:	Low	0.0V min, 0.5V max @ 8mA
	High	5.0V max, 2.7V min @ 400μA
J2:	Low	0.0V min, 0.45V max @ 1.5mA (J2)
	High	5.0V max, 2.4V min @ 100μA

Input voltage levels:

J1:	Low	0.0V min, 0.8V max
	High	2.0V min, 5.5V max
J2:	Low	-0.5V min, 0.8V max
	High	2.0V min, 5.5V max

Interrupt input: TTL-compatible
rising-edge triggered

Counter/Timer Section:

No. of counters:	3, 16 bits wide each
Type:	Presettable synchronous down counters
Maximum count frequency:	8MHz
Internal clock source:	PC clk÷2, PC clk÷4 (jumper selectable)

Miscellaneous:

Operating temperature:	0-60°C
Dimensions:	4.2" x 7.0"

*These specifications are derived from tests on a 12MHz 80286-based AT compatible. Maximum speed will vary depending on the speed of the host computer.

B. ESP-8680

Processor	14 MHz 5-volt "8680" (8086 equivalent)
Serial Port	RS-232
Graphics	CGA (Color Graphics Adapter) or LCD (Liquid Crystal Display)
Memory	256K x 8 EPROM or Flash Memory 512K or 1MB DRAM (8-bit or 16-bit wide memory path)
Memory Option	Expansion board adds up to 16MB DRAM
Bus Interface	ISA (Industry Standard Architecture)
Form Factor	1.7" x 5.2" (13.2cm x 4.3cm)
Power Consumption	Draws from 1 mA (sleep mode) to 300 mA (peak load powering back-lit LCD and peripherals)

C. MOTOROLA PVT6

Receiver Architecture	6-channel L1 1575.42 MHz
Tracking Capability	6 simultaneous satellite vehicles
Dynamics	Velocity 1000 Knots (514m/sec) Acceleration 4g
Aquisition Time (Time to First Fix, TTFF)	24 sec. typical TTFF (with current almanc, position, and time) 54 sec. typical TTFF
Accuracy	Less than 25 meters, SEP (without SA)
Operating Temperature	-30°C to +80°C
Physical Dimensions	3.94 x 2.75 x 0.65 inches (100 x 70 x 16.5 mm)
Weight	4.5 ounces (128 grams)
Switched Power	9-16 Vdc or 5 ± 0.25 Vdc
Keep-Alive Power	4.75 - 16 Vdc; 0.3 mA max
Power Consumption	Typical 1.3 W @ 5Vdc input 1.8 W @ 12Vdc input
MTBF	65,000 hours (estimated)

D. GYRATION GYROENGINE

Performance

motor speed: 8,000 revolutions per minute (RPM) to 50,000 RPM selectable by input voltage

Scorsby drift:

Temperature		25,000 RPM	50,000 RPM
-40°C	Typical Maximum	6.0°/min. 15.0°/min.	5.0°/min. 12.0°/min.
+25°C	Typical Maximum	2.0°/min. 5.0°/min.	1.5°/min. 3.0°/min.
+80°C	Typical Maximum	4.0°/min. 10.0°/min.	3.0°/min. 7.0°/min.

Static Drift

-40°C to +80°C; typical 0.5°/min., maximum 1.0°/min. at 25,000 to 50,000 RPM

Precession

erection system: pendulous inner gimbal

precession period: 135 seconds

precession damping: precession decreases 37% each precession period

precession angle due to horizontal acceleration: 3°/sec./G, equivalent to 0.14°/1 mile/hour change in speed

Optical Encoder Output

encoder resolution: 0.1°

encoder linearity: ±1% of full scale maximum

sensor slew rate: 260°/sec.

Power
   DC supply voltage: 4V to 12V selects motor speed
   maxium DC starting current:
      200 mA at 5V
      500 mA at 7V
      700 mA at 9V
   maximum DC running current:
      50 mA at 5V corresponds to 10,000 RPM
      125 mA at 7V corresponds to 25,000 RPM
      350 mA at 9V corresponds to 50,000 RPM

Environmental
   operational temperature range: -40°C t +80°C
   humidity: 80% relative
   pressure altitude: -1,000 ft. to 40,000 ft.
   shock resistance: 400 G's, 8 mS half sine wave
      pulse duration (all axes)
   vibration: 2 G's RMS (20 Hz to 2,000 Hz)

Physical
   dimensions: 1.3 in. maximum diameter
                1.75 in. maximum length
   weight: 1.5 oz.
   external: molded optical-grade polycarbonate with
      butyl rubber expansion seal
   internal: gimbal floatation fluid (Brayco 1721
      aerospace low-viscosity vacuum oil)

Service Life
   40,000 hours

E. KVH C100 DIGITAL COMPASS SENSOR

Accuracy	±0.5° or ±10 mils RMS
Repeatability	±0.2° or ±5 mils
Resolution	0.1° or 5 mils
Dip Angle	±80° Maintains stated accuracy after autocal over ±80° Magnetic Dip Angle
Tilt Angle	±16° Dev.= ±0.3° RMS ±45° Dev.= ±0.5° RMS
Electrical Power	Input Voltage: +8 to +20 VDC or +20 to +30 VDC (user selectable) Current Drain: 20 mA DC; nominal
Size	1.80 x 4.50 x 1.10" (4.6 cm x 11.4 cm x 2.8 cm)
Weight	2.0 ounces (57 grams)
Environmental Performance	Operating Temp.: -22°F to +122°F (-30°C to +50°C) Vibration: 30 minutes random MIL-STD-810 Shock: Handling shock per MIL-STD-810
Digital Interfaces	Standard RS232 Bidirectional Serial Data
Analog Outputs	Sine/Cosine: Sine/Cosine output voltage +2.5V±1.0V OR Linear Voltage: 0 to +3.6VDC into 10K Ohm minimum load

F. OMEGA PX176-100PSIS

PARAMETER	MIN	TYP	MAX	UNITS
Full Scale Output (FSO) @ 25°C	4.90	4.95-5.05	5.10	Vdc
Null Offset @ 25°C	.85	.95-1.05	1.15	Vdc
Linearity (Best Fit)		±.2	±.5	%FSO
Hysteresis		±.25		%FSO
Temperature Error Null 0° to 85°C		±.01	±.02	%FSO/°C
-55° to 0°C +85°C to 105°C		±.02		%FSO/°C
Sensitivity 0° to 85°C		±.01	±.02	%FSO/°C
-55° to 0°C +85°C to 105°C		±.02		%FSO/°C
Stability (1 year)		±1.0		%FSO
Frequency Response		10		kHz
Supply Voltage	9		20	Vdc
Supply Current (Quiecent)		15		mA

APPENDIX B

SOURCE CODE

```
--*****************************************************************
--     File Name : SANS.A
--     Author : Lcdr C. D. Stevens
--     DATE   : 12/3/92
--     Adapted from SNDL.a authored by Se-Hung Kwak
--     Comments : SANS consists of 3 major operations
--                AUV_MONITOR is a busy wait
--                  while other operations are idle
--                  AUV_MONITOR waits for a position update
--                  request
--                DR NAVIGATION consists of logging DEPTH, HEADING
--                  and PITCH along with the current time in small
--                  time increments for post-processing
--                INIT_PROCESS_GPS initializes the Motorola GPS
--                  and PROCESS_GPS_DATA logs all Motorola pro-
--                  prietary Binary Messages for Post-processing
--*****************************************************************

with NAV_DATA, CALENDAR, P_AUV, P_GPS, A_to_D, tty, text_io,
  gyro_cnt, MOTOROLA;
use  NAV_DATA, CALENDAR, P_AUV, P_GPS, A_to_D, tty, text_io,
  gyro_cnt, MOTOROLA;

procedure SANS is

  package INTEGER_INOUT is new INTEGER_IO(INTEGER);
  package FLOAT_INOUT is new FLOAT_IO(FLOAT);
  use INTEGER_INOUT, FLOAT_INOUT;


    GPS_DATA : NAV_DATA_TYPE;
    DIGITAL_HEADING, DIGITAL_DEPTH : INTEGER;
    HEADING, HDG_CHAN, DEPTH, DEPTH_CHAN, PITCH_COUNT : INTEGER;
    PITCH_DATA, HDG_DATA, DEPTH_DATA : FILE_TYPE;
    GPS_FILE : BYTE_FILE.FILE_TYPE;
    FIX_SECONDS : DURATION;
    FIX_TIME : TIME;
    GOOD_POSITION, UPDATE_REQUESTED : BOOLEAN := FALSE;
  begin
    BYTE_FILE.OPEN(GPS_FILE, MODE => BYTE_FILE.OUT_FILE, NAME =>
       "GPS.DAT");
    OPEN(PITCH_DATA, MODE => OUT_FILE, NAME => "PITCH.DAT");
    OPEN(HDG_DATA, MODE => OUT_FILE, NAME => "HDG.DAT");
    OPEN(DEPTH_DATA, MODE => OUT_FILE, NAME => "DEPTH.DAT");
    A_TO_D_PARAMETERS(HDG_CHAN, DEPTH_CHAN);
    INIT_PROCESS_GPS_DATA;
    INIT_PROCESS_AUV_DATA;
```

```
    INIT_DIGITAL_COUNTER;


    loop
      PROCESS_AUV_DATA(UPDATE_REQUESTED);
      CURRENT_DIGITAL_VALUE(HDG_CHAN, FIX_TIME, DIGITAL_HEADING);
      -- Convert Digital Heading to a compass reading
      HEADING := CURRENT_HEADING(DIGITAL_HEADING);
      FIX_TIME := CLOCK;
      FIX_SECONDS := SECONDS(FIX_TIME);
      PUT(HDG_DATA, HEADING);
      PUT(HDG_DATA, FLOAT(FIX_SECONDS));
      put(heading);
      text_io.put_line("heading");
      NEW_LINE(HDG_DATA);

      CURRENT_DIGITAL_VALUE(DEPTH_CHAN, FIX_TIME, DIGITAL_DEPTH);
      -- Convert Digital Depth to actual Depth
      DEPTH := CURRENT_DEPTH(DIGITAL_DEPTH);
      PUT(HDG_DATA, HEADING);
      PUT(HDG_DATA, FLOAT(FIX_SECONDS));
      put(heading);
      text_io.put_line("heading");
      NEW_LINE(HDG_DATA);

      PITCH_COUNT := READ_PITCH;
      PUT(PITCH_DATA, PITCH_COUNT);
      PUT(PITCH_DATA, FLOAT(FIX_SECONDS));
      put(pitch_count);
      text_io.put_line("pitch");
      NEW_LINE(PITCH_DATA);

      PROCESS_GPS_DATA(GPS_DATA, GOOD_POSITION, GPS_FILE);

      exit when GOOD_POSITION or tty.char_ready;

    end loop;
    BYTE_FILE.CLOSE(GPS_FILE);
    CLOSE(PITCH_DATA);
    CLOSE(HDG_DATA);
    CLOSE(DEPTH_DATA);
    SHUTDOWN_PROCESS_GPS_DATA;
    SHUTDOWN_PROCESS_AUV_DATA;
    RESTORE_INTERRUPTS;

end SANS;
```

```
--*******************************************************************
--
--    File Name : AUV_MON.A
--    Author    : Lcdr C. D. Stevens
--    DATE      : 5/15/93
--    Comments  : Adapted from P_GPS by Dr. Se-Hung Kwak to
--                  implement a serial line communication with an
--                  AUV which will request a GPS/Position Update
--*******************************************************************

-- Nav_data contains the declarations for the Nav Data record
format
package P_AUV is

  procedure init_Process_AUV_Data;

  procedure SHUTDOWN_Process_AUV_Data;

  procedure Process_AUV_Data(UPDATE_REQUESTED : in out BOOLEAN);

end P_AUV;

-- serial is an assembly language program which buffers incoming
-- serial stream data
with  TEXT_IO, SERIAL;
use   TEXT_IO, SERIAL;

package body P_AUV is

  LOCAL_TIME_DIFF, RECEIVER_TYPE_SIZE, PORT,
       BAUD,DATA_BIT,STOP_BIT: INTEGER;
  PARITY : CHARACTER;

  -- get parameters to establish serial connection
  procedure GET_AUV_PARAMETERS is
    TIME_ZONE_INFO, RECEIVER_INTERVAL_INFO, LINE : STRING(1..80);
    ZONE_INFO_SIZE, INTERVAL_SIZE,SIZE : INTEGER;
    INF : TEXT_IO.FILE_TYPE;
  begin
    OPEN(INF, MODE => IN_FILE, NAME => "AUVSETUP.DAT");
    GET_LINE(INF, LINE, SIZE);
    PORT := INTEGER'VALUE(LINE(1..SIZE));
    GET_LINE(INF, LINE, SIZE);
    BAUD := INTEGER'VALUE(LINE(1..SIZE));
    GET_LINE(INF, LINE, SIZE);
    DATA_BIT := INTEGER'VALUE(LINE(1..SIZE));
    GET_LINE(INF, LINE, SIZE);
    PARITY := LINE(1);
    GET_LINE(INF, LINE, SIZE);
    STOP_BIT := INTEGER'VALUE(LINE(1..SIZE));
    CLOSE(INF);
  end get_AUV_Parameters;
```

```
  procedure init_Process_AUV_data is
  begin
    get_AUV_Parameters;
    -- establish a serial connection
    -- and begin processing raw data
    open_serial(port,baud,data_bit,parity,stop_bit);
  end init_Process_AUV_data;


  procedure SHUTDOWN_Process_AUV_Data is
  begin
    close_serial;
  end SHUTDOWN_Process_AUV_Data;

  procedure Process_AUV_Data(UPDATE_REQUESTED : in out BOOLEAN)
is
    UNS_BYTE_INT : UNS8;
    FLAG : character;
  begin
    loop
      READ_CHAR(UNS_BYTE_INT, FLAG);
      exit when FLAG = 'Y';
    end loop;
    UPDATE_REQUESTED := TRUE;

    exception
      when others =>
       UPDATE_REQUESTED := FALSE;

  end PROCESS_AUV_DATA;


end P_AUV;
```

```
--*****************************************************************
--
--     File Name : P_GPS.A
--     Author : Se-Hung Kwak
--     DATE     : 9/11/91
--     MODIFIED : Lcdr C. D. Stevens
--     DATE     : 10/08/92
--     COMMENTS :  Init_Process_GPS reads a setup file for
--                   parameters to initialize a serial port
--                   (RS-232) connection in this version, actual
--                   data is read from a file
--                 Shutdown_Process_GPS terminates the serial
--                   connection
--                 Process_GPS_Data calls a procedure to
--                   initialize the proper receiver type and begin
--                   processing
--*****************************************************************

with NAV_DATA, TEXT_IO, MOTOROLA;
use  NAV_DATA, TEXT_IO, MOTOROLA;
package P_GPS is

  procedure init_Process_GPS_Data;

  procedure SHUTDOWN_Process_GPS_Data;

  procedure Process_GPS_Data(GPS_DATA : in out NAV_DATA_TYPE;
                      END_DATA : in out BOOLEAN;
                      GPS_FILE : in out BYTE_FILE.FILE_TYPE);
end P_GPS;

-- serial is an assembly language program which buffers
-- serial stream data
with  SERIAL, MOTOROLA, TEXT_IO, NAV_DATA;
use   SERIAL, MOTOROLA, TEXT_IO, NAV_DATA;

package body P_GPS is

  GPS_RECEIVER_TYPE : STRING(1..80);
  LOCAL_TIME_DIFF, RECEIVER_TYPE_SIZE, PORT,
     BAUD,DATA_BIT,STOP_BIT: INTEGER;
  PARITY : CHARACTER;

  -- get parameters to establish serial connection
  procedure GET_GPS_PARAMETERS is
    TIME_ZONE_INFO, RECEIVER_INTERVAL_INFO, LINE : STRING(1..80);
    ZONE_INFO_SIZE, INTERVAL_SIZE,SIZE : INTEGER;
    INF : TEXT_IO.FILE_TYPE;
```

```
  begin
    OPEN(INF, MODE => IN_FILE, NAME => "SETUP.DAT");
    GET_LINE(INF, GPS_RECEIVER_TYPE, RECEIVER_TYPE_SIZE);
    GET_LINE(INF, LINE, SIZE);
    PORT := INTEGER'VALUE(LINE(1..SIZE));
    GET_LINE(INF, LINE, SIZE);
    BAUD := INTEGER'VALUE(LINE(1..SIZE));
    GET_LINE(INF, LINE, SIZE);
    DATA_BIT := INTEGER'VALUE(LINE(1..SIZE));
    GET_LINE(INF, LINE, SIZE);
    PARITY := LINE(1);
    GET_LINE(INF, LINE, SIZE);
    STOP_BIT := INTEGER'VALUE(LINE(1..SIZE));

    -- time zone is not used in sans
    GET_LINE(INF, TIME_ZONE_INFO, ZONE_INFO_SIZE);
    LOCAL_TIME_DIFF :=
            INTEGER'VALUE(TIME_ZONE_INFO(1..ZONE_INFO_SIZE));
    CLOSE(INF);
  end get_GPS_Parameters;


  procedure init_Process_GPS_data is
  begin
    get_GPS_Parameters;
    -- establish a serial connection
    -- and begin processing raw data
    open_serial(port,baud,data_bit,parity,stop_bit);
  end init_Process_GPS_data;


  procedure SHUTDOWN_Process_GPS_Data is
  begin
    close_serial;
  end SHUTDOWN_Process_GPS_Data;


  procedure Process_GPS_DATA(GPS_DATA : in out NAV_DATA_TYPE;
                    END_DATA : in out BOOLEAN;
                    GPS_FILE : in out BYTE_FILE.FILE_TYPE) is
  begin
    GET_MOTOROLA_DATA(GPS_FILE, GPS_DATA, END_DATA);
    exception
      when others =>
     put_line("end3");
     END_DATA := TRUE;
 end Process_GPS_DATA;

end P_GPS;
```

```
--*****************************************************************
--     Title    : Motorola.a
--     Author   : Lcdr C. D. Stevens
--     DATE     : 9/11/91
--     Comments : GET_MOTOROLA_DATA records variable length
--                  strings of Motorola Proprietary Binary Data.
--                  It is linked with package SANS.
--                POST_PROCESS_MOTOROLA_DATA processes position
--                  format binary messages from an input file
--                  GPS.DAT and records them in a file OUTPUT.DAT.
--                  It is linked with package
--                  MOTOROLA_POST_PROCESSOR.
--                Package Serial declares a type UNS8 which is a
--                  byte_integer in the range 0..255 to be
--                  compatible with Motorola binary format
--                a problem exists in CHECK_SUM
--*****************************************************************

with  TEXT_IO, NAV_DATA, SERIAL, SEQUENTIAL_IO;
use   NAV_DATA, SERIAL;

package MOTOROLA is

  package BYTE_FILE is new SEQUENTIAL_IO(UNS8);
  use BYTE_FILE;

  -- procedure records data to a file for post-processing
  procedure GET_MOTOROLA_DATA(GPS_FILE : in out
                                       BYTE_FILE.FILE_TYPE;
                     GPS_DATA: out NAV_DATA_TYPE;
                     END_of_DATA :out BOOLEAN);

  -- procedure reads a file and performs conversions
  procedure POST_PROCESS_MOTOROLA_DATA(TEST_FILE: in out
                                         BYTE_FILE.FILE_TYPE;
                     OUTPUT_FILE: in out TEXT_IO.FILE_TYPE;
                     GPS_DATA: in out NAV_DATA_TYPE;
                     END_DATA : in out BOOLEAN);

end MOTOROLA;

with  text_io, SERIAL, MATH_LIB, BIT_OPS;
use  text_io, SERIAL, MATH_LIB, BIT_OPS;

package body MOTOROLA is
  package INTEGER_INOUT is new INTEGER_IO(INTEGER);
  package LONG_INTEGER_INOUT is new INTEGER_IO(LONG_INTEGER);
  package UNS_BYTE_INTEGER_INOUT is new INTEGER_IO(UNS8);
  package BYTE_INTEGER_INOUT is new INTEGER_IO(byte_integer);
  package FLOAT_INOUT is new FLOAT_IO(FLOAT);
  use INTEGER_INOUT, UNS_BYTE_INTEGER_INOUT, BYTE_INTEGER_INOUT,
            FLOAT_INOUT, LONG_INTEGER_INOUT;
```

```
-- Motorola's start frame chars are '@@'
-- end frame chars are 'Cr' and 'Lf'

FLAG : character;
temp : float;

procedure GET_MOTOROLA_DATA(GPS_FILE : in out
                                BYTE_FILE.FILE_TYPE;
                   GPS_DATA: out NAV_DATA_TYPE;
                   END_OF_DATA : out BOOLEAN) is

  UNS_BYTE_INT : UNS8;
begin
  loop
    READ_CHAR(UNS_BYTE_INT, FLAG);
    exit when FLAG = 'N';
    WRITE(GPS_FILE, UNS_BYTE_INT);

    -- *** DISPLAY OUTPUT FOR OBSERVATION PURPOSES ONLY ***
    PUT(UNS_BYTE_INT);

  end loop;
  exception
    when others =>
   END_OF_DATA := TRUE;

end GET_MOTOROLA_DATA;


procedure POST_PROCESS_MOTOROLA_DATA(TEST_FILE: in out
                                       BYTE_FILE.FILE_TYPE;
                         OUTPUT_FILE: in out TEXT_IO.FILE_TYPE;
                         GPS_DATA: in out NAV_DATA_TYPE;
                         END_DATA : in out BOOLEAN) is

  INT_LONG : LONG_INTEGER := 0;
  UNS_BYTE_INT : UNS8 := 0;
  CHECK_SUM : BYTE_INTEGER := 0;
  FIX_SECONDS : DURATION;
  COUNT, INT_PDOP : INTEGER := 0;
  POSIT_FORMAT : BOOLEAN := FALSE;
  TEMP_TIME : FLOAT;
```

```
function PROCESS_CHECK_SUM(UNS_BYTE_INT : UNS8)
                  return BYTE_INTEGER is
begin
  if UNS_BYTE_INT < 128 then
 CHECK_SUM := CHECK_SUM XOR BYTE_INTEGER(UNS_BYTE_INT);
  else
 -- remove MSB to bring UNS_BYTE_INT into range of byte
 -- integer then two's complement the result of XOR to
 -- account for MSB's effect
 if CHECK_SUM < 0 then
   CHECK_SUM := ((CHECK_SUM XOR
                   BYTE_INTEGER(UNS_BYTE_INT-128))+1)*(-1);
 else
   CHECK_SUM := (-1) * (CHECK_SUM XOR
                    BYTE_INTEGER(UNS_BYTE_INT-128))-1;
 end if;
  end if;
  return CHECK_SUM;
end PROCESS_CHECK_SUM;

procedure PROCESS_LONG_INTEGER(
                        INT_LONG      : in out LONG_INTEGER;
                        COUNT         : in out INTEGER) is
  TWOS_COMPLEMENT : BOOLEAN := FALSE;
  BYTE_INT : BYTE_INTEGER;
  TEMP_LONG_INT : LONG_INTEGER;

  procedure GET_NEXT_BYTE is
  begin
    BYTE_FILE.READ(TEST_FILE, UNS_BYTE_INT);
    CHECK_SUM := PROCESS_CHECK_SUM(UNS_BYTE_INT);
    COUNT := COUNT + 1;
    if TWOS_COMPLEMENT then
      TEMP_LONG_INT := LONG_INTEGER(UNS_BYTE_INT) - 255;
    else
      TEMP_LONG_INT := LONG_INTEGER(UNS_BYTE_INT);
    end if;
  end GET_NEXT_BYTE;
begin
  if UNS_BYTE_INT > 127 then
    TEMP_LONG_INT := LONG_INTEGER(UNS_BYTE_INT) - 255;
    TWOS_COMPLEMENT := TRUE;
  else
    TEMP_LONG_INT := LONG_INTEGER(UNS_BYTE_INT);
    TWOS_COMPLEMENT := FALSE;
  end if;
  INT_LONG := TEMP_LONG_INT * 2**24;
  GET_NEXT_BYTE;
  INT_LONG := INT_LONG + (TEMP_LONG_INT * 2**16);
  GET_NEXT_BYTE;
  INT_LONG := INT_LONG + (TEMP_LONG_INT * 2**8);
  GET_NEXT_BYTE;
```

```
  INT_LONG := INT_LONG + TEMP_LONG_INT;
end PROCESS_LONG_INTEGER;

procedure  PROCESS_TIME(TEMP_TIME     : in out FLOAT;
                COUNT         : in out INTEGER) is
begin
  TEMP_TIME := 0.0;
  BYTE_FILE.READ(TEST_FILE, UNS_BYTE_INT);
  CHECK_SUM := PROCESS_CHECK_SUM(uns_byte_int);
  TEMP_TIME := FLOAT(UNS_BYTE_INT) * 3600.0;
  COUNT := COUNT + 1;
  TEMP_TIME := TEMP_TIME + float(UNS_BYTE_INT) * 60.0;
  BYTE_FILE.READ(TEST_FILE, UNS_BYTE_INT);
  CHECK_SUM := PROCESS_CHECK_SUM(uns_byte_int);
  COUNT := COUNT + 1;
  TEMP_TIME := TEMP_TIME + float(UNS_BYTE_INT);
  PROCESS_LONG_INTEGER(INT_LONG, COUNT);
  TEMP_TIME := TEMP_TIME + float(INT_LONG) * (10.0 ** (-9));
  PUT(OUTPUT_FILE, ' ');
  PUT(OUTPUT_FILE, TEMP_TIME, fore => 8, aft => 2, exp => 0);
  PUT(OUTPUT_FILE, ' ');
end PROCESS_TIME;

procedure  PROCESS_VELOCITY(
                 COUNT         : in out INTEGER) is
  VEL, VEL_N, VEL_E, HDG, RADIANS : float := 0.0;
begin
  VEL := FLOAT(UNS_BYTE_INT) * FLOAT(2 ** 8);
  BYTE_FILE.READ(TEST_FILE, UNS_BYTE_INT);
  CHECK_SUM := PROCESS_CHECK_SUM(uns_byte_int);
  COUNT := COUNT + 1;
  vel := (vel + float(uns_byte_int))/100.0;
  BYTE_FILE.READ(TEST_FILE, UNS_BYTE_INT);
  CHECK_SUM := PROCESS_CHECK_SUM(uns_byte_int);
  COUNT := COUNT + 1;
  HDG := FLOAT(UNS_BYTE_INT) * FLOAT(2 ** 8);
  BYTE_FILE.READ(TEST_FILE, UNS_BYTE_INT);
  CHECK_SUM := PROCESS_CHECK_SUM(uns_byte_int);
  COUNT := COUNT + 1;
  HDG := (HDG + float(uns_byte_int))/10.0;
  RADIANS := HDG/57.2958;
  VEL_N := COS(RADIANS) * VEL;
  VEL_E := SIN(RADIANS) * VEL;
  put(output_file, vel_n, fore => 5, aft => 2, exp => 0);
   PUT(OUTPUT_FILE, ' ');
  put(output_file, vel_e, fore => 5, aft => 2, exp => 0);
   PUT(OUTPUT_FILE, ' ');
  put(output_file, hdg, fore => 5, aft => 2, exp => 0);
   PUT(OUTPUT_FILE, ' ');
  put(output_file, vel, fore => 5, aft => 2, exp => 0);
   PUT(OUTPUT_FILE, ' ');
end PROCESS_VELOCITY;
```

```
procedure  PROCESS_PDOP(
                           COUNT          : in out INTEGER) is
  PDOP : float := 0.0;
begin
--    GPS_DATA.PDOP := INTEGER(UNS_BYTE_INT) * (2**8);
  PDOP := FLOAT(UNS_BYTE_INT) * FLOAT(2 ** 8);
  BYTE_FILE.READ(TEST_FILE, UNS_BYTE_INT);
  CHECK_SUM := PROCESS_CHECK_SUM(uns_byte_int);
  PDOP := (PDOP + float(UNS_BYTE_INT))/10.0;
  PUT(OUTPUT_FILE, PDOP, fore => 4, aft => 0, exp => 0);
--    GPS_DATA.PDOP := (GPS_DATA.PDOP +
                           INTEGER(UNS_BYTE_INT))/10;
  COUNT := COUNT + 1;
end PROCESS_PDOP;

procedure SYNCH_WITH_HEADER is
  FIRST_DELIMITER : BOOLEAN := FALSE;
begin
  -- loop until ASCII character @ is received (first
  -- delimiter)
  loop
    BYTE_FILE.READ(TEST_FILE, UNS_BYTE_INT);
    PUT(OUTPUT_FILE, COUNT);
    exit when UNS_BYTE_INT = 64;
  end loop;
  BYTE_FILE.READ(TEST_FILE, UNS_BYTE_INT);
  PUT(OUTPUT_FILE, COUNT);
  if UNS_BYTE_INT = 64 then
    BYTE_FILE.READ(TEST_FILE, UNS_BYTE_INT);
    PUT(OUTPUT_FILE, COUNT);
    if UNS_BYTE_INT = 66 then
      BYTE_FILE.READ(TEST_FILE, UNS_BYTE_INT);
      PUT(OUTPUT_FILE, COUNT);
      if UNS_BYTE_INT = 97 then
        POSIT_FORMAT := TRUE;
        COUNT := 4;
        CHECK_SUM := 35;
      end if;
    end if;
  end if;
end SYNCH_WITH_HEADER;
```

```
  begin
    SYNCH_WITH_HEADER;
    loop
      BYTE_FILE.READ(TEST_FILE, UNS_BYTE_INT);
      COUNT := COUNT + 1;
      PUT(OUTPUT_FILE, COUNT);
      CHECK_SUM := PROCESS_CHECK_SUM(uns_byte_int);

      if POSIT_FORMAT then
       if count = 9 then
       -- get time in seconds
       PROCESS_TIME(TEMP_TIME, COUNT);

     -- get latitude
     elsif COUNT = 16 then
       PROCESS_LONG_INTEGER(INT_LONG, COUNT);
       temp := float(int_long)/(3.6 * (10.0 ** 6));
       put(output_file, temp, fore => 6, aft => 6, exp => 0);
       PUT(OUTPUT_FILE, ' ');
--          GPS_DATA.LATITUDE := NAV_CONVERSION(INT_LONG);

     -- get longitude
     elsif COUNT = 20 then
       PROCESS_LONG_INTEGER(INT_LONG, COUNT);
       temp := float(int_long)/(3.6 * (10.0 ** 6));
       put(output_file, (360.0 + temp), fore => 6, aft => 6, exp
            => 0);
       put(output_file, ' ');
--          GPS_DATA.LONGITUDE := NAV_CONVERSION(INT_LONG);

     -- get ellipsoidal height
     elsif COUNT = 24 then
       PROCESS_LONG_INTEGER(INT_LONG, COUNT);
       temp := float(int_long)/(100.0);
       put(output_file, temp, fore => 6, aft => 2, exp => 0);
       PUT(OUTPUT_FILE, ' ');

     elsif COUNT = 32 then
       PROCESS_VELOCITY(COUNT);

     elsif COUNT = 36 then
       PROCESS_PDOP(COUNT);

     elsif COUNT = 65 then
       if CHECK_SUM = BYTE_INTEGER(UNS_BYTE_INT) then
         PUT(OUTPUT_FILE, " GOOD CHECKSUM");
       else
         PUT(OUTPUT_FILE, " BAD CHECKSUM");
       end if;
```

```
    elsif UNS_BYTE_INT = 13 then
      -- count is 65 vice 67
      BYTE_FILE.READ(TEST_FILE, UNS_BYTE_INT);
      COUNT := COUNT + 1;
      if UNS_BYTE_INT = 10 then
        exit;
      end if;
    end if;
     end if;
   end loop;
   new_line(output_file);

   --NAV_DATA.PUT(GPS_DATA);

   exception
     when others =>
    put_line("end of file");
    END_DATA := TRUE;
  end POST_PROCESS_MOTOROLA_DATA;

end MOTOROLA;
```

```
--*****************************************************************
--
--    File Name : SERIAL_D.A
--    Author : Se-Hung Kwak
--    DATE   : 9/11/91
--    Modified  : Lcdr C. D. Stevens to include type UNS8 which
--                   is an unsigned byte integer in the range
--                   0..255 to be
--                   compatible with Motorola Proprietary Binary
--                   Format
--*****************************************************************

package SERIAL is
  -- UNS8 is a subtype byte_integer (8-bit integer)
  -- which is natural vice 2's complement
  type UNS8 is range 0..255;

--  procedure READ_CHAR(CH, DATA_READY : out CHARACTER);
  --  pragma INTERFACE(assembly, read_char);

--  function WRITE_CHAR(CH: in CHARACTER) return CHARACTER;
  --  pragma INTERFACE(assembly, write_char);

  procedure READ_CHAR(CH : out UNS8;
                  DATA_READY : out CHARACTER);
    pragma INTERFACE(assembly, read_char);

  function WRITE_CHAR(CH: in CHARACTER) return UNS8;
    pragma INTERFACE(assembly, write_char);

  procedure OPEN_SERIAL(PORT, BAUD, DATA_BIT: INTEGER;
                PARITY:CHARACTER; STOP: INTEGER);


  procedure
INIT_SERIAL(RX_REG,TX_REG,INT_EN,LINE_CRT,MODEM_CRT,LINE_STAT,
               BAUD_LSB,LINE,INT_MASK,INT_NUM: in INTEGER);
    pragma INTERFACE(assembly, init_serial);


  procedure CLOSE_SERIAL;
    pragma INTERFACE(assembly, close_serial);

end SERIAL;
```

```
--*****************************************************************
--
--     File Name : SERIAL.A
--     Author : Se-Hung Kwak
--     DATE   : 9/11/91
--
--*****************************************************************

package body SERIAL is

  procedure OPEN_SERIAL(PORT, BAUD, DATA_BIT: INTEGER;
                  PARITY:CHARACTER; STOP: INTEGER) is
    OFFSET : constant INTEGER := 16#100#;
    RX_REG : INTEGER := 16#2F8#;
    TX_REG : INTEGER := 16#2F8#;
    INT_EN : INTEGER := 16#2F9#;
    LINE_CRT  : INTEGER := 16#2FB#;
    MODEM_CRT : INTEGER := 16#2FC#;
    LINE_STAT : INTEGER := 16#2FD#;
    BAUD_LSB, LINE, INT_MASK, INT_NUM : INTEGER;
  begin
    if PORT = 1 then
      RX_REG := RX_REG + OFFSET;
      TX_REG := TX_REG + OFFSET;
      INT_EN := INT_EN + OFFSET;
      LINE_CRT  := LINE_CRT + OFFSET;
      MODEM_CRT := MODEM_CRT + OFFSET;
      LINE_STAT := LINE_STAT + OFFSET;
    end if;                                  -- default port2

    if BAUD = 1200 then
      BAUD_LSB := 96;
    elsif BAUD = 2400 then
      BAUD_LSB := 48;
    elsif BAUD = 4800 then
      BAUD_LSB := 24;
    else
      BAUD_LSB := 12;                       -- default port2
    end if;

    if DATA_BIT = 5 then
      LINE := 0;
    elsif DATA_BIT = 6 then
      LINE := 1;
    elsif DATA_BIT = 7 then
      LINE := 2;
    else
      LINE := 3;                           -- default 8 data bits
    end if;
```

```
    if STOP = 2 then
      LINE := LINE + 16#4#;
    end if;                                  -- default 1 stop bit

    if PARITY /= 'N' then
      LINE := LINE + 16#8#;
      if PARITY = 'E' then
     LINE := LINE + 16#10#;
      end if;                                -- default parity : odd
    end if;

    if PORT = 1 then
      INT_MASK := 16#EF#;                    -- reset bit 4
    else
      INT_MASK := 16#F7#;                    -- default port2 & reset bit
3
    end if;

    if PORT = 1 then
      INT_NUM := 16#0C#;                     -- SERIAL 1 INTERRUPT #, 0CH
(12)
    else
      INT_NUM := 16#0B#;                     -- default SERIAL2 INT #, 0BH
(11)
    end if;

INIT_SERIAL(RX_REG,TX_REG,INT_EN,LINE_CRT,MODEM_CRT,LINE_STAT,
          BAUD_LSB,LINE,INT_MASK,INT_NUM);

  end OPEN_SERIAL;

end SERIAL;

;*****************************************************************
;
;   File Name : SERIAL.ASM
;   Author : Se-Hung Kwak
;   DATE   : 9/11/91
;   Modified  : Lcdr C. D. Stevens to eliminate the stripping of
;                  bit 7 from ASCII characters since Motorola
;                  proprietary binary format uses all 8 bits
;
;*****************************************************************

     NAME   SERIAL
     DGROUP  GROUP  DATA

data    segment para public 'data'
bufsiz  equ     4096
buffer  db      bufsiz dup(0)              ; buffer
bufptrl dw      0                          ; points to start of
```

```
buffer
bufptr2 dw      0                               ; points to end of buffer
bufcs   dw      0                               ; interrupt vector cs
buffer
bufip   dw      0                               ; interrupt vector ip
buffer

RX_REG   DW     0                               ; RX REG ADDRESS
TX_REG   DW     0                               ; TX REG ADDRESS
INT_EN   DW     0                               ; INT ENABLE REG ADDRESS
LINE_CRT  DW    0                               ; LINE CRT REG ADDRESS
MODEM_CRT DW    0                               ; MODEM CRT REG ADDRESS
LINE_STAT DW    0                               ; LINE STAT REG ADDRESS
BAUD_LSB  DW    0                               ; BAUD DIVISOR (LSB)
LINE      DW    0                               ; LINE VALUE (DATA
BIT,PARITY,STOP)
INT_MASK  DW    0                               ; 8259 INT MASK VALUE FOR
PORT1 OR 2
INT_NUM   DW    0                               ; INTERRUPT NUMBER FOR
PORT1 OR 2
data    ends
;

_SERIAL  segment para public 'code'
     ASSUME CS:_SERIAL, DS:DGROUP


;*****************************************************************
******
;
; Procedure INIT_SERIAL(RX_REG,TX_REG,INT_EN,LINE_CRT,MODEM_CRT,
;                       LINE_STAT,BAUD_LSB,LINE,INIT_MASK,
;                       INT_NUM : in INTEGER);
;
;*****************************************************************
******


     PUBLIC  INIT_SERIAL
```

```
INIT_SERIAL   PROC  FAR
;
     cli                         ;disable all interrupts
     PUSH       BP
     MOV        BP, SP
     PUSH       DS
     MOV        AX, DATA
     MOV        DS, AX
     PUSH       DX
     PUSH       DI
     PUSH       SI
     PUSH       CX
     mov        ax, [BP+6]        ; GET RX_REG ADDR
     mov        RX_REG, ax
     mov        ax, [BP+8]        ; GET TX_REG ADDR
     mov        TX_REG, ax
     mov        ax, [BP+10]       ; GET INT_EN ADDR
     mov        INT_EN, ax
     mov        ax, [BP+12]       ; GET LINE_CRT ADDR
     mov        LINE_CRT, ax
     mov        ax, [BP+14]       ; GET MODEM_CRT ADDR
     mov        MODEM_CRT, ax
     mov        ax, [BP+16]       ; GET LINE_STAT ADDR
     mov        LINE_STAT,ax
     mov        ax, [BP+18]       ; GET BAUD_LSB
     mov        BAUD_LSB,ax
     mov        ax, [BP+20]       ; GET LINE
     mov        LINE, ax
     mov        ax, [BP+22]       ; GET INT_MASK
     mov        INT_MASK, ax
     mov        ax, [BP+24]       ; GET INT_NUM
     mov        INT_NUM, ax
;
;  set baud
;
     mov        dx, LINE_CRT      ; select baud divisor
     mov        ax, dx
     mov        al, 80h
     out        dx, al
     mov        dx, RX_REG        ; LSB divisor
     mov        ax, BAUD_LSB
     out        dx, al
     mov        dx, INT_EN        ; MSB divisor
     mov        al, 0
     out        dx, al
;
; init line control reg.
;
     mov        dx, LINE_CRT
     mov        ax, LINE
     out        dx, al
```

```
;
; init modem control reg.
;
    mov         dx, MODEM_CRT
    mov         al, 0Bh             ; loop back test
    out         dx, al
;
; enable interrupts
;
    mov         dx, INT_EN
    mov         al, 1               ; enabled receiver-data-ready
    out         dx, al
;
;   save interrupt vector
;
    push        es              ; es:bx vector will be returned
    push        bx
    mov         ax, INT_NUM ; give interrupt number
    mov         ah, 35h         ; dos function call #35h: get vector
    int         21h             ; dos function call int 21h
    mov         bufip, bx       ; save ip
    mov         bufcs, es       ; save cs
    pop         bx
    pop         es
    mov         cx, INT_NUM ; save INT_NUM into CX because of DS
;                                 change
;
;   Set up interrupt vector table
;
    push        ds              ; ds:dx will be saved into vector
table
    push        dx
    mov         ax, offset asyint
    mov         dx, ax
    mov         ax, cs
    mov         ds, ax
    mov         ax, CX          ; give interrupt number
    mov         ah, 25h         ; dos function call #25h: Set int
                                ; vector
    int         21h             ; dos function call int 21h
    pop         dx
    pop         ds
```

```
;
;  adjust interrupt mask reg in 8259
;
     in          al, 21h      ; interrupt mask pattern
     and         ax, INT_MASK ; enable irq3 or 4 by resetting
                              ; proper bit
     out         21h, al      ; save to interrupt mask reg in 8259

     POP         CX
     POP         SI
     POP         DI
     POP         DX
     POP         DS
     POP         BP
     sti
     RET         20
INIT_SERIAL   ENDP

;*****************************************************************
;
; Procedure CLOSE_SERIAL
;
;*****************************************************************

     PUBLIC  CLOSE_SERIAL

CLOSE_SERIAL   PROC  FAR
     cli                      ;disable all interrupts
     PUSH        DS
     MOV         AX, DATA
     MOV         DS, AX
     PUSH        CX
;
;  adjust interrupt mask reg in 8259
;
     push        bx
     mov         bx, INT_MASK ; get INT_MASK pattern
     not         bx           ; flip INT_MASK pattern
     mov         ax, bx
     in          al, 21h      ; interrupt mask pattern
     or          ax, bx       ; disable irq3 or 4 by setting
                              ; proper bit
     out         21h, al      ; save to interrupt mask reg in 8259
     pop         bx

     MOV         CX, INT_NUM  ; save INT_NUM into CX because of DS
                              ;  change
```

```
;
; restore  interrupt vector for serial-2
;
     push       ds            ; ds:dx will be saved into vector
                              ; table
     push       dx
     mov        dx, bufip
     mov        ds, bufcs
     mov        ax, CX        ; get proper INTERRUPT NUMBER
     mov        ah, 25h       ; dos function call #25h: Set int
                              ;  vector
     int        21h           ; dos function call int 21h
     pop        ds
     pop        dx

     POP        CX
     POP        DS
     sti                      ; enable all interrupts
     RET
CLOSE_SERIAL   ENDP

;*******************************************************************
;
; Procedure READ_CHAR(CH, DATA_READY : out CHARACTER);
;                     DATA_READY = 'Y' New CH
;                     DATA_READY = 'N' NO  CH
;*******************************************************************

     PUBLIC  READ_CHAR
READ_CHAR   PROC   FAR
     STI
     PUSH       BP
     MOV        BP, SP
     PUSH       DS
     MOV        AX, DATA
     MOV        DS, AX
     call       chget
     mov        bx, ax                 ; save received char
     mov        al, ah
     PUSH       ES
     LES        SI, DWORD PTR [BP+10]
     MOV        AL, 'N'
     MOV        ES:[SI], AL            ; DATA_READY = N
     CMP        AH, 0
     JE         R_END                  ; NO Ch Available -> return
     LES        SI, DWORD PTR [BP+10]
     MOV        AL, 'Y'
     MOV        ES:[SI], AL            ; YES, ch. DATA_READY = Y
     LES        SI, DWORD PTR [BP+6]
     mov        ax, bx                 ; restore received char
```

```
     MOV         ES:[SI], AL          ; Return CH
R_END:  POP         ES
     POP         DS
     POP         BP
     RET         8
READ_CHAR   ENDP

;*******************************************************************
;
; Function WRITE_CHAR(CH: in CHARACTER) return CHARACTER
;                     Return 'Y'  CH is out
;                     Return 'N'  CH is not out. Buffer is full
;*******************************************************************

     PUBLIC  WRITE_CHAR
WRITE_CHAR   PROC    FAR
     STI
     PUSH        BP
     MOV         BP, SP
     PUSH        DS              ; Save DS
     MOV         AX, DATA        ; Data is accessable
     MOV         DS, AX
     MOV         CL, 'N'         ; TX buf is full
     MOV         DX, LINE_STAT     ; Line Status Reg
     IN          AL, DX
     TEST        AL, 20H         ; TX is empty?
     JZ          W_END           ; Not empty, return
     MOV         AL, [BP+6]      ; Get CHAR
     MOV         DX, TX_REG      ; Output to TX Reg
     OUT         DX, AL
     MOV         CL, 'Y'         ; Success
W_END:
     POP         DS
     POP         BP
     RET         2

WRITE_CHAR  ENDP

;
; serial communication interrupt routine
;
asyint  proc     far
     push    dx
     push    bx
     push    ax
```

```
;
; place the ascii char into the buffer.
;
     cli
     push    ds
     mov     ax, data
     mov     ds, ax
     mov     dx, RX_REG                ; read data port
     in      al,dx
; DO NOT strip bit 7 for Motorola unsigned byte integer
operations
;        and     al,7fh                   ; strip off bit 7
     mov     bx,bufptr2                ; bx <- bufptr2
     mov     [buffer+bx], al           ; save into buffer
     inc     bx                        ; inc ptr2
     cmp     bx,bufsiz                 ; end of buffer ?
     jc      asyskip                   ; no
     mov     bx,0                      ; yes, wrap around
asyskip: cmp    bx,bufptr1                ; buffer full ?
     jz      end_asy                   ; yes, ignore input data
     mov     bufptr2,bx                ; save ptr2 into bufptr2
;
end_asy: mov    al,20h                    ; send EOI (end of
interrupt) command
     out     20h,al                    ; to port 20 (8259 command
reg)
     pop     ds
     sti
     pop     ax
     pop     bx
     pop     dx
     iret
asyint  endp
;
;
;
; get character (al <- data,  ah <- 1 : success, ah <- 0 : buffer
empty)
;
chget   proc        near
     push       bx
;        cli                              ; disable all interrputs
     mov        bx,bufptr1             ; get ptr1
     cmp        bx,bufptr2             ; buffer empty ?
     jnz        chget2                 ;
     mov        ah,0                   ; no char in the buffer
     jmp        chgete                 ; get out from chget
;
chget2: mov        al,[buffer+bx]        ; NO, pass char through
                                         ; al reg
     inc        bx                     ; inc ptr1
     cmp        bx, bufsiz             ; end of buffer ?
```

```
     jc          chget3                   ;
     mov         bx,0                     ; YES, reset ptrl
chget3: mov         bufptrl,bx              ; save ptrl
     mov         ah,1                     ; success
chgete:
     ;sti                                  ; enable int
     pop         bx
     ret
chget   endp

;
;
;
; display a char on the screen in the al reg with ascii format
;
;
disply proc       near
     push        bx
     push        ax              ; save char

;
; prepare to display the char.
;
     mov         bx,0
     mov         ah,14
     int         10h
     pop         ax
     push        ax
     cmp         al,0dh
     jnz         end_dis
;
; return -> return + line feed
;
     mov         al,0ah
     mov         bx,0
     mov         ah,14
     int         10h
end_dis: pop        ax
     pop         bx
     ret
disply endp
;

_SERIAL   ends
     end
```

```
--*******************************************************************
--
--   File Name : ANALOG.A
--   Author    : Lcdr C. D. Stevens
--   Date      : 01 December 1992
--   Comments  : A_to_D is a package which controls the operation
--                 of the Sapphire A to D converter through I/O
--                 reads and writes
--               procedures are provided to read the ports from a
--                 setup file and eventually the base address
--
--*******************************************************************

with TEXT_IO, CALENDAR;
use TEXT_IO, CALENDAR;
package A_to_D is

  procedure A_TO_D_PARAMETERS(HDG_PORT, DEPTH_PORT : out
                                                    INTEGER);

  procedure CURRENT_DIGITAL_VALUE(PORT     : in INTEGER;
                                  FIX_TIME : in out TIME;
                                  OUTPUT   : in out INTEGER);

  function CURRENT_HEADING(DATA : INTEGER) return INTEGER;

  function CURRENT_DEPTH(DATA : INTEGER) return INTEGER;

end A_to_D;
```

```
with PORT, SPIO, TEXT_IO, CALENDAR, SYSTEM;
use  PORT, SPIO, TEXT_IO, CALENDAR;
package body A_to_D is
  package INTEGER_INOUT is new INTEGER_IO(INTEGER);
  package FLOAT_INOUT is new FLOAT_IO(FLOAT);
  use INTEGER_INOUT;
  use FLOAT_INOUT;

  procedure A_TO_D_PARAMETERS(HDG_PORT, DEPTH_PORT :out INTEGER)
                                                              is
    INFO : FILE_TYPE;
    LINE : STRING(1..5);
    SIZE : INTEGER;
  begin
    -- open the setup file and read ports for depth and heading
    -- analog inputs (adjustable base address may be implemented
   -- later)
    OPEN(INFO, MODE => IN_FILE, NAME => "A_TO_D.dat");
    GET_LINE(INFO, LINE, SIZE);
    HDG_PORT := INTEGER'VALUE(LINE(1..SIZE));
    GET_LINE(INFO, LINE, SIZE);
    DEPTH_PORT := INTEGER'VALUE(LINE(1..SIZE));
  --  BASE_ADDR := INTEGER'VALUE(LINE(1..SIZE));
    CLOSE(INFO);
  end A_TO_D_PARAMETERS;

  procedure CURRENT_DIGITAL_VALUE(PORT     : in INTEGER;
                                  FIX_TIME : in out TIME;
                                  OUTPUT   : in out INTEGER) is

    IN_4LSB      : INTEGER := 16#300#; -- Input 4 Least
                                       --Significant Bits
                           -- Base 300 + Offset 0

    TRIGGER      : INTEGER := 16#301#; -- 12 Bit A to D
                                       -- Trigger(Output)
    IN_8MSB      : INTEGER := 16#301#; -- Input 8 MSB's
                                          -- Base 300 + Offset 1

    CONTROL_REG1 : INTEGER := 16#302#; -- Control Register 1
    STATUS_REG : INTEGER := 16#302#; -- A to D Conversion Status
                                     -- Register (In)
                           -- Base 300 + Offset 2

    CONTROL_REG2  : INTEGER := 16#303#; -- Control Register 2
                          -- Base 300 + offset 3
    VOLT_RNG      : INTEGER := 16#10#;  -- Write to Control
                                        -- Register 2
                          -- Analog Input Range 0-10 volts
    DATA_LSB, DATA_MSB, LOOPS : INTEGER := 0;

    procedure CONVERSION_STATUS is
```

```
  STATUS_VAL : INTEGER;
  CONVERSION_COMPLETE : BOOLEAN := FALSE;
begin
  -- check the status register until A to D conversion
  -- complete
  -- bit 7 = 0 indicates conversion complete
  loop
    STATUS_VAL := IN_BYTE(STATUS_REG);
    -- Hex 80 = 1000_0000 and STATUS_VAL is binary
    if STATUS_VAL < 16#80# then
      CONVERSION_COMPLETE := TRUE;
    end if;
    exit when CONVERSION_COMPLETE;
  end loop;
end CONVERSION_STATUS;
```

```
begin
  OUT_BYTE(CONTROL_REG2, VOLT_RNG);   -- write voltage range
  OUT_BYTE(CONTROL_REG1, PORT);       -- Select Mux Channel
  delay 0.001;                        -- wait for new channel to
                                      -- settle
  OUT_BYTE(TRIGGER, 1);               -- Trigger Conversion

  -- check status register and wait until conversion complete
  CONVERSION_STATUS;

  DATA_LSB := IN_BYTE(IN_4LSB);             -- Read LSB's
  DATA_MSB := IN_BYTE(IN_8MSB);             -- Read MSB's

  -- extract 12-bit integer from the two-byte code
  OUTPUT := (DATA_LSB/16) + (DATA_MSB * 16);
end CURRENT_DIGITAL_VALUE;

-- Convert the A to D output to a compass heading in degrees
-- 0.1 volts = 000 degrees
-- 1.9 volts = 360 degrees
function CURRENT_HEADING(DATA : INTEGER) return INTEGER is
  VOLTS, HEADING : FLOAT;
begin
  VOLTS := ((FLOAT(DATA) * 10.0)/4096.0);
  HEADING := (VOLTS - 0.1) * 200.0;
  return INTEGER(HEADING);
end CURRENT_HEADING;

-- Convert the A to D digital output to actual depth
-- 1.0 volts = 0 Feet
-- 6.0 volts = 220 Feet (@ 32 feet per atmosphere)
-- 0 - 100 PSIS (6.8 Atmospheres)
function CURRENT_DEPTH(DATA : INTEGER) return INTEGER is
  VOLTS, DEPTH : FLOAT;
begin
  VOLTS := (FLOAT(DATA) * 10.0)/4096.0;
  --convert to meters 9.85 meters/atmosphere
  DEPTH := (VOLTS - 1.0) * 9.85 * 6.8/5.0;
  return INTEGER(DEPTH);
end CURRENT_DEPTH;

end A_to_D;
```

```
--*******************************************************************
--    Title    : gyro_cnt.a
--    Author   : Se-Hung Kwak, Lcdr C. D. Stevens
--    DATE     : 2/20/93
--    Comments : Gryo_cnt.a provides the Pragma Interface with
--               assembly language Gyro_cnt.asm
--*******************************************************************

package GYRO_CNT is

  procedure INIT_DIGITAL_COUNTER;
    pragma INTERFACE (assembly, INIT_DIGITAL_COUNTER);

  function READ_PITCH return integer;
    pragma INTERFACE (assembly, READ_PITCH);

  procedure RESTORE_INTERRUPTS;
    pragma INTERFACE (assembly, RESTORE_INTERRUPTS);

  procedure SAPPHIRE_DELAY;
    pragma INTERFACE (assembly, SAPPHIRE_DELAY);

end GYRO_CNT;
```

```
;*****************************************************************
;
;   File Name : GYRO_CNT.ASM
;   Authors   : Se-Hung Kwak, LCDR C. D. STEVENS
;   DATE      : 2/20/93
;   Comments  : Implements an assembly language driver for a
;                 Gyration, Inc. miniature spin gyroscope
;               The service is interrupt driven through LPT1
;                 interfaced through a Sapphire A to D converter
;*****************************************************************

     NAME   GYRO_CNT
     DGROUP  GROUP  DATA

data    segment para public 'data'
;  PITCH_CALC variables
old_data  db      0                         ; previous state value
old_hi    db      0                         ; previous hi_bit value
old_lo    db      0                         ; previous lo_bit value
count     dw      0
bufip     dw      0
bufcs     dw      0
bufim     dw      0

; INTERRUPT INITIALIZATION variables
INT_MASK  DW    0                         ; 8259 INT MASK VALUE FOR
PORT1 OR 2
INT_NUM   DW    0                         ; INTERRUPT NUMBER FOR
PORT1 OR 2

data    ends
;

CODE  segment para public 'code'
     ASSUME CS:CODE, DS:DATA

;*************************************************************
;
; Procedure INIT_DIGITAL_COUNTER
;
;*************************************************************

     PUBLIC  INIT_DIGITAL_COUNTER

INIT_DIGITAL_COUNTER   PROC  FAR
;
     cli                         ;disable all interrupts
     PUSH       BP
     MOV        BP, SP
     PUSH       DS
     MOV        AX, DATA
     MOV        DS, AX
```

```
      PUSH        DX
      PUSH        DI
      PUSH        SI
      PUSH        CX
      mov         INT_MASK, 07fh     ; 8259 interrupt mask pattern
      mov         INT_NUM, 0fh      ; interrupt number for LPT1
;
; program sapphire board
;
      push        ax
      push        dx
      mov         dx,0302h
      mov         al,08
      out         dx,al     ; output port #302h 1000 (8h)
      pop         dx
      pop         ax
;
; set counter initial value
;
      mov         ax,0
      mov         count,ax
;
; save interrupt vector
;
      push        es            ; es:bx vector will be returned
      push        bx
      mov         ax, INT_NUM ; give interrupt number
      mov         ah, 35h       ; dos function call #35h: get vector
      int         21h           ; dos function call int 21h
      mov         bufip, bx     ; save ip
      mov         bufcs, es     ; save cs
      pop         bx
      pop         es
      mov         cx, INT_NUM ; save INT_NUM into CX because of DS
                                ; change
;
;   Set up interrupt vector table
;
      push        ds            ; ds:dx will be saved into vector
                                ; table
      push        dx
      mov         ax, offset PITCH_CALC
      mov         dx, ax
      mov         ax, cs
      mov         ds, ax
      mov         ax, CX        ; give interrupt number
      mov         ah, 25h       ; dos function call #25h: Set int
                                ; vector

      int         21h           ; dos function call int 21h
      pop         dx
      pop         ds
```

```
;
;  adjust interrupt mask reg in 8259
;
     in          al, 21h      ; interrupt mask pattern
     mov         bufim,ax     ; save interrupt mask for
                              ; restoration
     and         ax, INT_MASK ; enable irq5 by resetting properbit
     out         21h, al      ; save to interrupt mask reg in 8259

     POP         CX
     POP         SI
     POP         DI
     POP         DX
     POP         DS
     POP         BP
     sti
     RET

INIT_DIGITAL_COUNTER  ENDP
;
;*****************************************************************
;
;   function READ_PITCH return INTEGER;
;
;*****************************************************************
     PUBLIC  READ_PITCH
read_pitch   proc   far
;
     sti
     push        ds
     mov         ax,data
     mov         ds,ax
     mov         cx,count   ; recall current pitch_count
     pop         ds
     ret
read_pitch   endp
;
```

```
;*****************************************************************
;
; PITCH_CALC interrupt service routine
;
;
;*****************************************************************
;
;
;
      PUBLIC     PITCH_CALC
PITCH_CALC   PROC    FAR
     CLI
     PUSH       DX          ; save register contents
     PUSH       AX
     PUSH       DS          ;  make data segment visible
     MOV        AX,DATA
     MOV        DS,AX

;
;  read  status register  and evaluate state changes
;
     mov        dx,0302h
     in         al,dx          ; input al register contents of
status register
     mov        ah,al
;        and        ah,07h        not needed but may affect
output pattern
;                                 OP4 thru OP1

     and        al,30h          ; extract bits 5 and 6
     cmp        al,30h         ; test new case for state 11
     jne        update         ; no other case matters
     mov        dl,old_data    ; recall last state for comparisons

     cmp        dl,10h          ; check for cw rotation
     jne        ccw

     inc        count
     jmp        update

ccw:    cmp        dl,20h          ; check for ccw rotation
     jne        update
     dec        count

update: mov        old_data,al    ; update previous state value in
                                ; data

     MOV        AL,20H         ; send EOI (end of interrupt)
                               ; command
     OUT        20H,AL         ; to port 20 (IBM specific - 8259
                               ; register)
     mov        dx,302h        ; reset int
```

```
     mov        al,ah
     or         al,08h
     out        dx,al

     POP        DS              ; restore data segment
     POP        AX              ; restore registers
     POP        DX
     STI
     IRET                       ; return control to BP addr
(previous routine)
                         ; (interrupt return)
PITCH_CALC  ENDP
;
;
;
;******************************************************************
;
;   procedure RESTORE_INTERRUPTS (termination routine)
;
;******************************************************************
     PUBLIC  RESTORE_INTERRUPTS
restore_interrupts   proc   far
;
     cli
     push       dx            ; save registers
     push       ds
     mov        ax,data       ; make data visible
     mov        ds,ax

     mov        ax,bufim       ; recall interrupt mask
     out        21h, al        ; restore to interrupt mask reg
     mov        dx,bufip      ; recall initial interrupt vector
     mov        ax,bufcs
     mov        ds,ax
     mov        ax,0fh        ; load interrupt level
     mov        ah, 25h       ; dos function call #25h: Set int
                              ; vector
     int        21h           ; dos function call int 21h

     pop        ds            ; restore registers
     pop        dx
     sti
     ret
restore_interrupts   endp
;
```

```
;
; Debugging routine
; display a char on the screen in the al reg with ascii format
; example :
;        mov       ax,[some register]
;        call      disply
;
;
disply  proc       near
     push      bx
     push      ax             ; save char

;
; prepare to display the char.
;
     mov       bx,0
     mov       ah,14
     int       10h
     pop       ax
     push      ax
     cmp       al,0dh
     jnz       end_dis
;
; return -> return + line feed
;
     mov       al,0ah
     mov       bx,0
     mov       ah,14
     int       10h
end_dis:pop       ax
     pop       bx
     ret
disply endp
;
CODE   ends
     end
```

```
--*****************************************************************
--
--    File Name : NAV.A
--    Author    : Lcdr C.D. Stevens
--    Date      : 12-3-92
--    Comments  : Nav_Data provides the declarations for a
--                  navigation data type
--                function nav conversion converts
--                  a long_integer input to a type degree type
--                Put outputs a nav_data_type
--*****************************************************************

with CALENDAR;  use CALENDAR;
package NAV_DATA is

  type DEGREE_TYPE is
    record
      DEGREES   : INTEGER;
      MINUTES   : INTEGER;
      SECONDS   : FLOAT;
    end record;

  type NAV_DATA_TYPE is
    record
      FIX_TIME  : TIME;
      LATITUDE  : DEGREE_TYPE;
      LONGITUDE : DEGREE_TYPE;
      DEPTH     : FLOAT;
      PDOP      : INTEGER;
    end record;

  -- Data type to track AUV's ascent vector
  type DELTA_TYPE is
    record
      DELTA_LAT   : FLOAT;
      DELTA_LONG  : FLOAT;
    end record;

  function NAV_CONVERSION(VAR  : LONG_INTEGER;
                DATA : DEGREE_TYPE) return DEGREE_TYPE;

  procedure PUT(DATA : in NAV_DATA_TYPE);

end NAV_DATA;
```

```
with TEXT_IO, CALENDAR;
use TEXT_IO, CALENDAR;
package body NAV_DATA is
  package INTEGER_INOUT is new INTEGER_IO(INTEGER);
  package FLOAT_INOUT is new FLOAT_IO(FLOAT);
  use INTEGER_INOUT, FLOAT_INOUT;

  -- 648,000,000 = 180 DEGREES (longitude)
  -- range of latitude is +/- 324,000,000 (+/- 90 degrees)
  function NAV_CONVERSION(VAR  : LONG_INTEGER;
                DATA : DEGREE_TYPE) return DEGREE_TYPE is
    RESULT : DEGREE_TYPE;
    DEG, MIN, REMAINDER  : LONG_INTEGER;
  begin
    -- integer division to extract degree component
    DEG := VAR/360000;
    RESULT.DEGREES := INTEGER(DEG);
    -- integer division to extract minute component from
    -- remainder
    MIN := (VAR - DEG * 360000)/6000;
    RESULT.MINUTES := INTEGER(MIN);
    REMAINDER := VAR - ((DEG * 360000) + (MIN * 6000));
    -- float division to extract seconds component
    RESULT.SECONDS := FLOAT(REMAINDER)/100.0;
    return RESULT;
  end NAV_CONVERSION;

  procedure PUT(DATA : in NAV_DATA_TYPE) is
    SECS : DURATION;
  begin
    if DATA.LATITUDE.DEGREES >= 0 then
      PUT('N');
    else
      PUT('S');
    end if;
    PUT(DATA.LATITUDE.DEGREES, WIDTH => 4);
    PUT(" DEG ");
    PUT(DATA.LATITUDE.MINUTES, WIDTH => 3);
    PUT(" MIN ");
    PUT(DATA.LATITUDE.SECONDS, FORE => 3, AFT => 2, EXP => 0);
    PUT_LINE(" SEC");

    if DATA.LONGITUDE.DEGREES >= 0 then
      PUT('E');
    else
      PUT('W');
    end if;
    PUT(DATA.LONGITUDE.DEGREES, WIDTH => 4);
    PUT(" DEG ");
    PUT(DATA.LONGITUDE.MINUTES, WIDTH => 3);
    PUT(" MIN ");
    PUT(DATA.LONGITUDE.SECONDS, FORE => 3, AFT => 2, EXP => 0);
```

```
    PUT_LINE(" SEC ");

    PUT("DEPTH : ");
    PUT(DATA.DEPTH, FORE => 3, AFT => 2, EXP => 0);
    NEW_LINE;
    PUT("PDOP : ");
    PUT(DATA.PDOP, WIDTH => 5);
    NEW_LINE;
    PUT("TIME : ");
    SECS := SECONDS(DATA.FIX_TIME);
    PUT(FLOAT(SECS), FORE => 6, AFT => 2, EXP => 0);
    NEW_LINE;
  end PUT;

end NAV_DATA;
```

```
--
******************************************************************
--
--   File Name : DR.A
--   Author    : Lcdr C. D. Stevens
--   Date      : 03 December 1992
--   Comments  : DR provides the basic navigation functions
--               Delta_Update calculates the latitude and
--                 longitude change
--                 based on the heading, climb angle and depth
--                 change since
--                 last delta_update and maintains a cumulative
--                 vector
--
--
******************************************************************
with  NAV_DATA, DELTA_POSIT;
use  NAV_DATA, DELTA_POSIT;
package DR is

  procedure DELTA_UPDATE(PITCH_ANGLE    : in FLOAT;
                         DEPTH_CHANGE   : in FLOAT;
                         HEADING        : in FLOAT;
                         DELTA_POS      : in out DELTA_TYPE);

end DR;
```

```
with NAV_DATA, DELTA_POSIT, MATH_LIB, TEXT_IO;
use NAV_DATA, DELTA_POSIT, MATH_LIB, TEXT_IO;
package body DR is


  -- not yet operational as a post proccessor
  -- can be linked with operational SANS for real-time processing
  -- of DR_Navigation data
  procedure DELTA_UPDATE(PITCH_ANGLE    : in FLOAT;
                DEPTH_CHANGE   : in FLOAT;
                HEADING        : in FLOAT;
                DELTA_POS      : in out DELTA_TYPE) is

    TWO_PI : FLOAT := 6.2832;
    DELTA_LATIT, DELTA_LONGIT : FLOAT;
    ADJACENT, DISTANCE, RADIANS : FLOAT;
  begin
    -- Calculate the distance travelled by examining the pitch
angle
    -- and depth change to calculate the horizontal component
    ADJACENT := SIN(PITCH_ANGLE)/COS(PITCH_ANGLE);   --Tan not
found
    DISTANCE := DEPTH_CHANGE/ADJACENT;

    -- Determine the latitude and longitude components of the
distance
    RADIANS := HEADING/TWO_PI;
    DELTA_LATIT := DISTANCE/COS(RADIANS);
    DELTA_LONGIT := DISTANCE/SIN(RADIANS);

    -- update the cumulative vector with the respective
components
    -- of latitude and longitude change
    DELTA_POS.DELTA_LAT := DELTA_POS.DELTA_LAT + DELTA_LATIT;
    DELTA_POS.DELTA_LONG := DELTA_POS.DELTA_LONG + DELTA_LONGIT;
  end DELTA_UPDATE;

end DR;
```

APPENDIX C

INTERRUPT ROUTINE MAJOR OPERATIONS

The development of the interrupt-driven routine for tracking gyro rotation was outlined in Section IV.B.2.f.(3). This appendix provides specific details outlining the implementation's four basic procedures.

A. INITIALIZE THE INTERRUPT ROUTINE

At the execution of the main procedure, the host machine's interrupt vector table must be adjusted. This permits the Sapphire interrupt input channel to trigger the Gyro's service routine.

1. The initialization routine, INIT_DIGITAL_COUNTER disables interrupts and saves the current value of registers to be used during initialization.

2. Interrupt enable is programmed in the Sapphire board through bit 3 (hexadecimal value 8) of control register 1 (base offset + 2). Interrupt enable must be reprogrammed after each interrupt by the interrupt service routine. The value of the pitch count is inititalized to 0.

3. The interrupt vector table is set to respond to hardware interrupt level 7 (IRQ7) by transferring control to the interrupt service routine PITCH_CALC. The interrupt number and effective address to which control must pass is added to the current vector table.

4. The current hardware level 7 interrupt vector is saved to be restored after termination of SANS. The interrupt level to be replaced (0F hexadecimal corresponds to IRQ7) is loaded into the lower half of the AX register (AL) and the function number (35 hexadecimal for retrieve current vector) is loaded into the upper half of the AX register (AH). AX contains the parameters for Disk Operating System (DOS) function call INT 21. The result of the call is the current code segment address (CS) returned to the ES register and program counter (IP) returned to the BX register (ES:BX). These results are stored in the data segment for later restoration. [Ref. 20]

5. After the current vector for LPT1 has been saved, the new vector is loaded into the interrupt vector table. This vector enables the interrupt service routine to be accessed at each interrupt. The offset operator is used to determine the offset address of the service routine PITCH_CALC. This address is loaded as the new address for the service routine in the interrupt vector table. To program the vector, the offset value is loaded in the DX register and the code segment address (CS register) is loaded in the DS register as parameters for DOS function call INT 21. Interrupt level (IRQ7) is loaded in AL as before, and DOS function number 25 hexadecimal is loaded in AH as additional parameters for INT 21. The result is a new service address for IRQ7 of DS:DX. [Ref. 20]

6. For IBM compatible architecture, the interrupt mask must be set in the 8259 programmable interrupt mask. The current interrupt mask is read from I/O port 21 hexadecimal. The value read from I/O port 21 is AND'ed with 7F hexadecimal (0111

1111 binary), to reset IRQ7 (bit 7). This operation and is performed on the current value and 7F hexadecimal to preserve all current interrupts while enabling the new interrupt. The result is written to port 21 hexadecimal.

7. The contents of the registers are then restored, interrupts re-enabled and control restored to the address in the base pointer.

B. IMPLEMENT THE SERVICE ROUTINE

1. After the interrupt vector table has been set, interrupts are handled by the interrupt service routine. The current state of both inner gimbals input signals are evaluated at each interrupt by reading the contents of the status register from Sapphire base offset + 2.

2. The values of bits 5 and 6 are extracted by operation and with 30 hexadecimal (0011 0000 binary). This makes all bits zero except bits 5 and 6 which retain their value. The result indicates the current state.

3. This state is compared with the previous state recalled from memory to determine if the state transition is valid as described by Figure 4.4. The cumulative rotation since initialization is incremented/decremented as appropriate.

4. The interrupt service routine resets interrupt enable in the Sapphire board through bit 3 of control register 1 for subsequent interrupts.

C. CURRENT_VALUE

The main procedure must be able to access the result of the interrupt service routine's tracking of the gyro rotation. Ada function READ_PITCH interfaces with

the assembly language procedure READ_PITCH which loads the current value of the pulse count from the data segment into register CX. The contents of CX is the function return value.

Whenever the gyro's CURRENT_VALUE is required by the main procedure, the pulse count (cumulative rotation since initialization) is determined by a function call to READ_PITCH. The difference since the last update is applied as gyro rotation to determine the new pitch angle.

D. RESTORE INTERRUPTS

Before program termination, the interrupt vector table is restored to its initial condition through procedure RESTORE_INTERRUPTS. This requires restoring the interrupt mask and vector stored in the data segment at initialization to their initial configuration.

1. Interrupts are disabled, register contents are saved and the data segment is made visible.

2. The original interrupt mask is loaded from memory to the AX register. This register's contents are written to port 21 to reconfigure the 8259.

3. The original interrupt vector table is restored by loading the original interrupt vector address for IRQ7 from memory into the DX:DX register pair. IRQ7, which is 0F hexadecimal, is loaded in AL as before and DOS function number 25 hexadecimal is loaded in AH as AX register parameters for DOS function call INT 21. The result of the DOS operation is restoration of the original service address for LPT1. [Ref. 20]

APPENDIX D

STATE TRANSITION TRACKING ALGORITHM VERIFICATION

CASE 1 : clockwise rotation (the counter is incrementing).

The result should be 4 units of rotation @ 0.4 degrees/unit.

IG-A	IG-B	PITCH COUNT	STATE	TRANSITION CASE
0	1	0	1	-
1	1	1	2	1
0	1	1	1	2
1	1	2	2	1
0	1	2	1	2
1	1	3	2	1
0	1	3	1	2
1	1	4	2	1

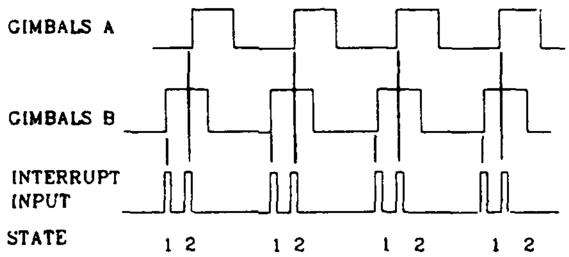


Figure D.1 : Case 1 State Transitions

CASE 3 : counter-clockwise rotation (the counter is decrementing).

The result should be -4 units from initial value.

IG-A	IG-B	PITCH COUNT	STATE	TRANSITION CASE
1	0	4	3	-
1	1	3	2	3
1	0	3	3	2
1	1	2	2	3
1	0	2	3	2
1	1	1	2	3
1	0	1	3	2
1	1	0	2	3

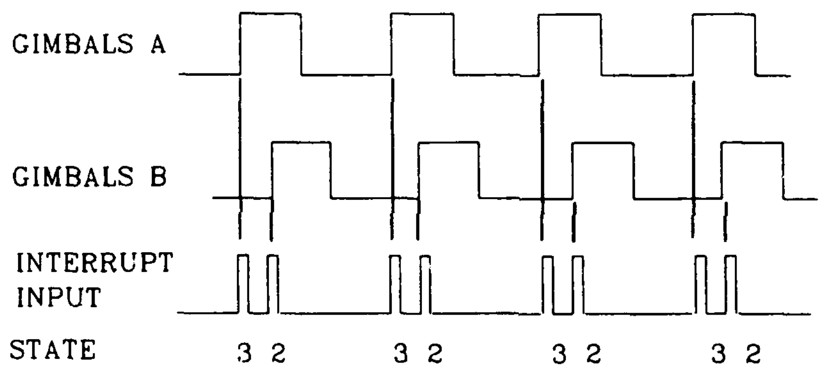


Figure D.2 : Case 3 State Transitions

CASE 1 to CASE 3 : reversal in rotation, clockwise to counter-clockwise.

The result should be 0, increment 2 then decrement 2.

IG-A	IG-B	PITCH COUNT	STATE	TRANSITION CASE
0	1	0	1	-
1	1	1	2	1
0	1	1	1	2
1	1	2	2	1
1	0	2	3	2
1	1	1	2	3
1	0	1	3	2
1	1	0	2	3

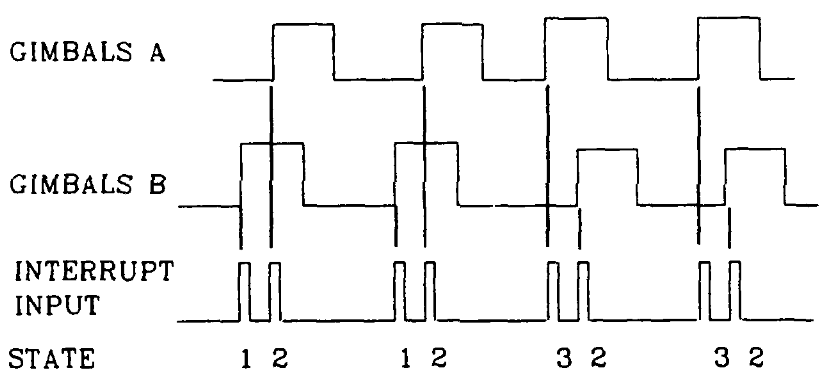


Figure D.3 : Case 1 to Case 3 Rotation Reversal

CASE 3 to CASE 1 : Reversal in rotation, counter-clockwise to clockwise.

The result should be 0, decrement 2 then increment 2.

IG-A	IG-B	PITCH COUNT	STATE	TRANSITION CASE
1	0	0	3	-
1	1	-1	2	3
1	0	-1	3	2
1	1	-2	2	3
0	1	-2	1	2
1	1	-1	2	1
0	1	-1	1	2
1	1	0	2	1

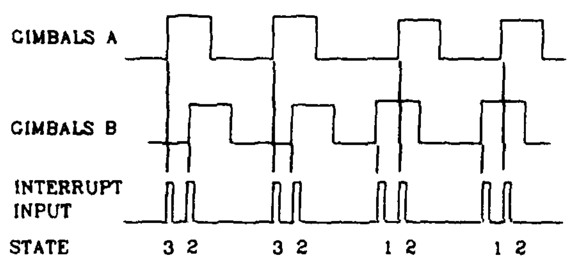


Figure D.4 : Case 3 to Case 1 Rotation Reversal

CASE 0 : Gyro is stationary with noise in IG-A.

The result should be 0 (ignore noise).

IG-A	IG-B	PITCH COUNT	STATE	TRANSITION CASE
1	0	0	3	-
1	0	0	3	0
1	0	0	3	0
1	0	0	3	0
1	0	0	3	0
1	0	0	3	0
1	0	0	3	0
1	0	0	3	0

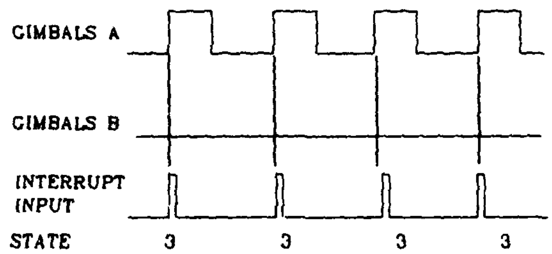


Figure D.5 : Noise in IG-A

CASE 0 : Gyro is stationary with noise in IG-B.

The result should be 0 (ignore noise).

IG-A	IG-B	PITCH COUNT	STATE	TRANSITION CASE
0	1	0	1	-
0	1	0	1	0
0	1	0	1	0
0	1	0	1	0
0	1	0	1	0
0	1	0	1	0
0	1	0	1	0
0	1	0	1	0

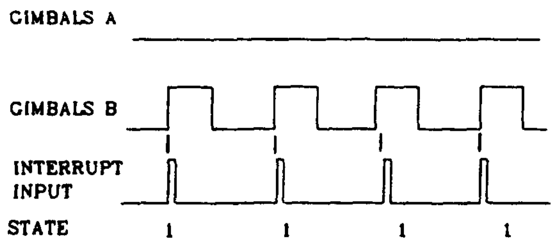


Figure D.6 : Noise in IG-B

CASE 0 : Gyro is stationary with noise in both channels.

The result should be 0 (ignore noise).

IG-A	IG-B	PITCH COUNT	STATE	TRANSITION CASE
1	1	0	2	-
1	1	0	2	0
1	1	0	2	0
1	1	0	2	0
1	1	0	2	0
1	1	0	2	0
1	1	0	2	0
1	1	0	2	0

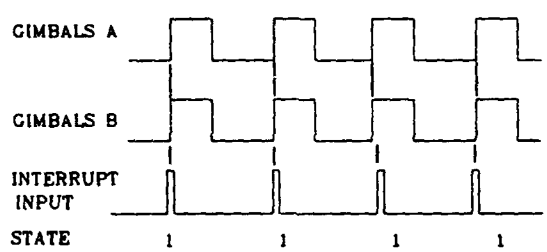


Figure D.7 : Noise in Both Channels

CASE 4 : Gyro is stationary with noise in both channels.

The result should be 0 (ignore noise).

IG-A	IG-B	PITCH COUNT	STATE	TRANSITION CASE
1	0	0	3	-
0	1	0	1	4
1	0	0	3	4
0	1	0	1	4
1	0	0	3	4
0	1	0	1	4
1	0	0	3	4
0	1	0	1	4

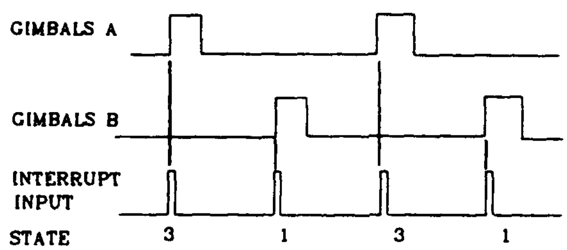


Figure D.8 : Noise in Both Input Channels

APPENDIX E

A. GPS MESSAGE TRANSLATION TEST CASES

The following is a typical 68 byte binary message with each byte represented by its equivalent integer value :

64	64	66	97				Header
1	26	7	201				Date
19	39	6	0	14	12	23	Time
7	218	54	110				Latitude
229	217	60	97				Longitude
0	0	3	84				Velocity
0	0	14	231				Heading
0	139	11	235				Height
0	73	0					PDOP/Type
4	4						Satellites/Tracked
15	8	96	128				Channel 1
25	8	112	136				Channel 2
0	0	0	0				Channel 3
29	8	107	136				Channel 4
14	8	112	136				Channel 5
0	0	0	0				Channel 6
32							Receiver Status
143							Checksum
13	10						End Delimiters

TEST CASE I

```
Header:
@      @      B      a      (This header corresponds to a
64     64     66     97      position format message and is
                             processed after storage)

Month  Day        Year (7 * 256 + 201 =1993)
1       26     7   201      =>   (January 26, 1993)

Hours/Minutes/Seconds and fractional seconds
19     39     6             0     14     12     23
                       =>   19:39:06.000920599

Latitude (in milliseconds)
 7                218               54               110
(7 * 2exp24)  + (218 * 2exp16) + (54 * 2exp8) + 110
                            =>   131,741,294

Longitude (in milliseconds)
229    217    60     97     =>   3,856,216,233 > 2exp31
(3,856,216,233 - 2exp32)  => -438,748,063 (two's complement)
```

Velocity	Heading	Height
0 0 3 84	0 0 14 231	0 139 11 235

PDOP (decimal value)	PDOP type
0 73	0

(0 * 2exp8) + 73 = 73

Satellite status information
4 (visible) 4 (tracked)

ID	mode	strength	flags
15	8	96	128
25	8	112	136
0	0	0	0
29	8	107	136
14	8	112	136
0	0	0	0

32 (receiver status) 143 (checksum)

13 10 <CR><LF> end of message delimiters

OUTPUT:
North 36.594803 degrees (131,741,294 milliseconds)
West 121.87446 degrees (-438,748,063 milliseconds)
PDOP : 7.3 (in decimal range 0.0 to 99.9 for Motorola)

TEST CASE II

Header corresponds to a satellite range format message. The message is not processed after writing to non-volatile memory.

@	@	B	g
64	64	66	103

Time (in seconds)	Fractional seconds (in nanoseconds)
2 12 66	1 15 90 225

SVID	Mode	GPS Time (Seconds/Fractional Seconds)	Carrier/Code Phase
18	8	8 12 92 66 225 8 212	82 12 4 8 12 9 11 7
13	8	8 12 92 66 225 8 210	88 11 5 6 66 8 22 7
8	8	8 12 92 66 225 8 224	99 13 6 8 88 9 11 8
9	8	8 12 92 66 225 8 214	103 12 4 5 99 8 33 9
24	0	7 16 88 55 107 0 112	122 55 5 8 88 9 22 6
16	8	8 12 92 66 225 8 214	44 66 7 8 55 8 11 6

73 Checksum (Exclusive OR of all bits after @@)
13 10 <CR><LF> (Carriage Return/Line Feed) End Delimiter

TEST CASE III

Only the latitude/longitude conversions are addressed henceforth.

Input:

Latitude (bytes 15-18)
255 218 54 125 (-2,476,329 milliseconds in two's complement)

Longitude (bytes 19-22)
255 217 60 72
(-2,440,472 milliseconds in two's complement)

Output:

South -0.687869 degrees
West -0.677909 degrees

TEST CASE IV

Input:

Latitude
0 0 0 3 (000,000,003 milliseconds)

Longitude
0 0 0 4 (000,000,004 milliseconds)

Output:

North 0.0000000 degrees
East 0.0000000 degrees

APPENDIX F

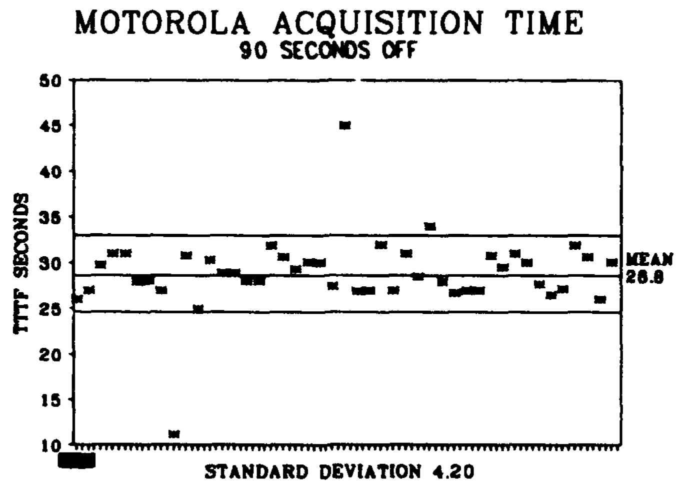


Figure F.1 : Motorola Acquisition Time After 90 Seconds Off

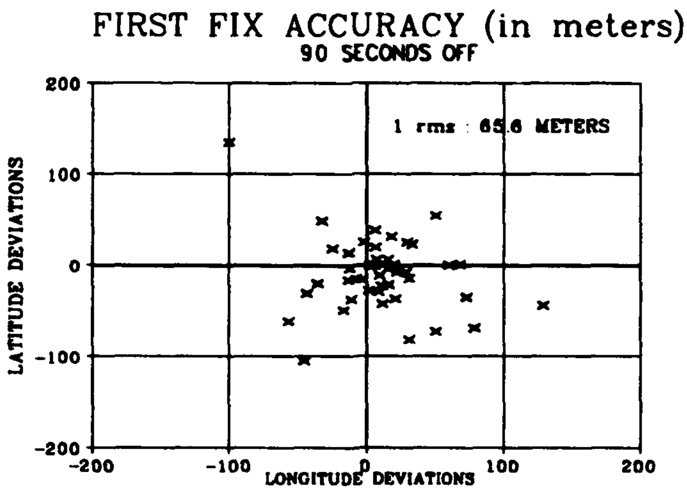


Figure F.2 : Motorola First Fix Accuracy After 90 Seconds Off

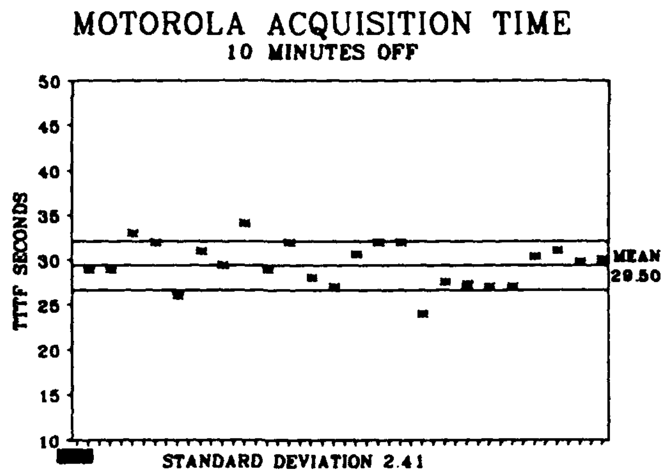


Figure F.3 : Motorola Acquisition Time After 10 Minutes Off

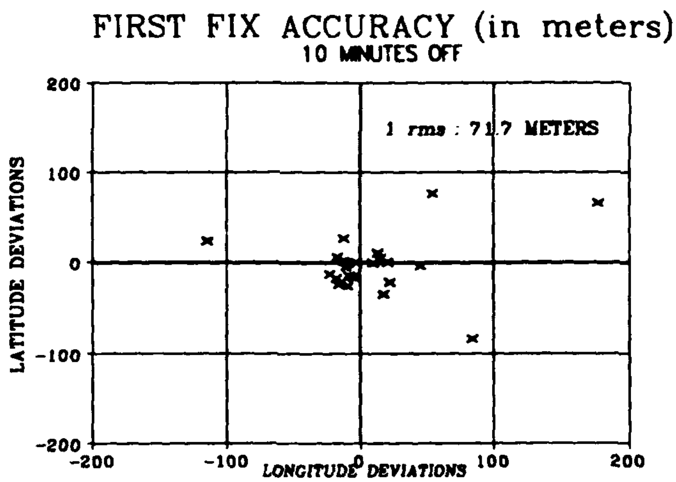


Figure F.4 : Motorola First Fix Accuracy After 10 Minutes Off

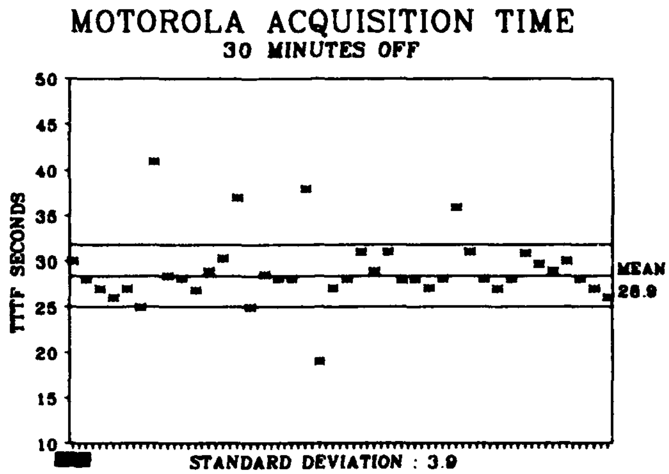


Figure F.5 : Motorola Acquisition Time After 30 Minutes Off

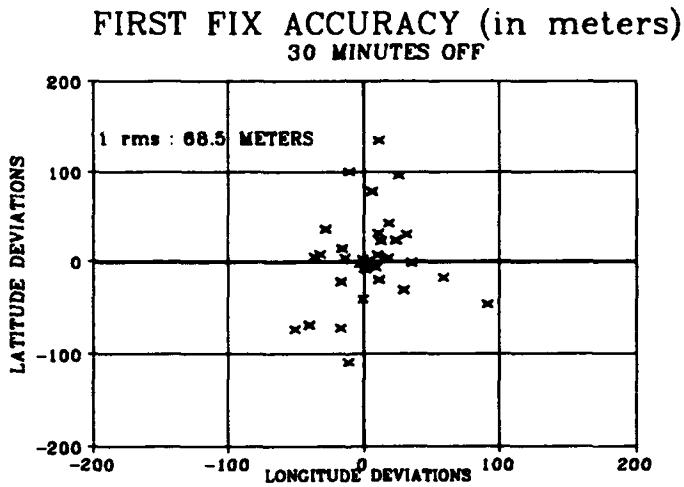


Figure F.6 : Motorola First Fix Accuracy After 30 Minutes Off

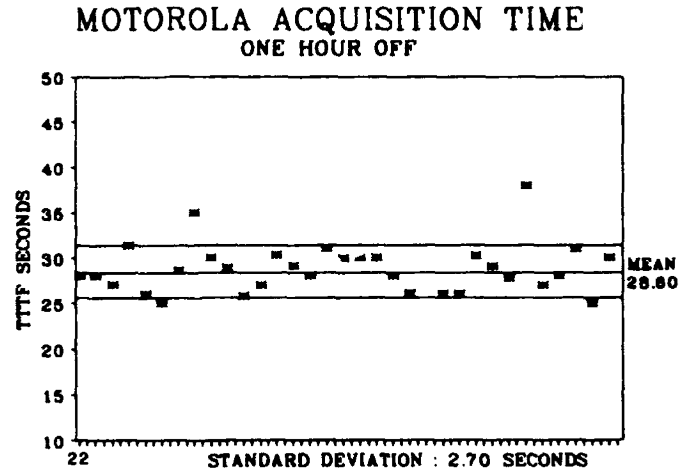


Figure F.7 : Motorola Acquisition Time After One Hour Off

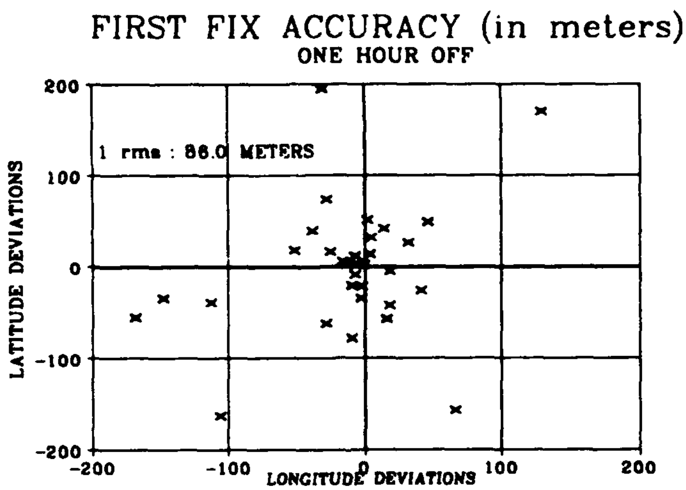


Figure F.8 : Motorola First Fix Accuracy After One Hour Off

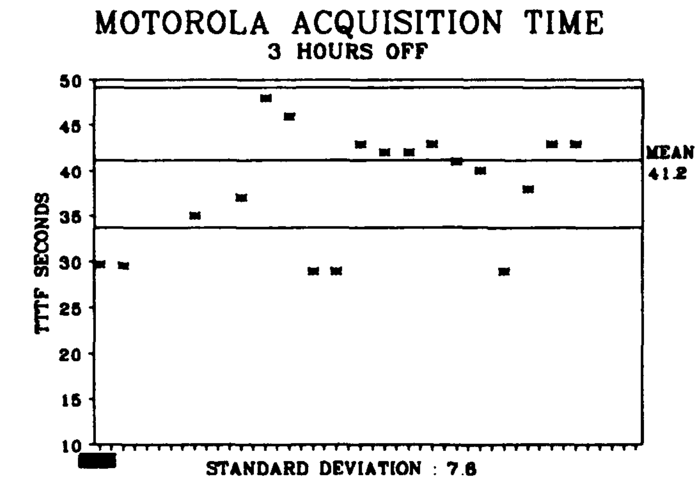


Figure F.9 : Motorola Acquisition Time After 3 Hours Off

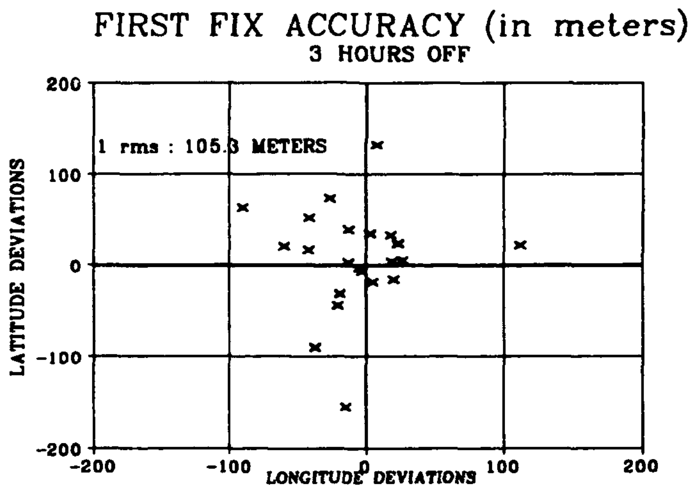


Figure F.10 : Motorola First Fix Accuracy After Three Hours Off

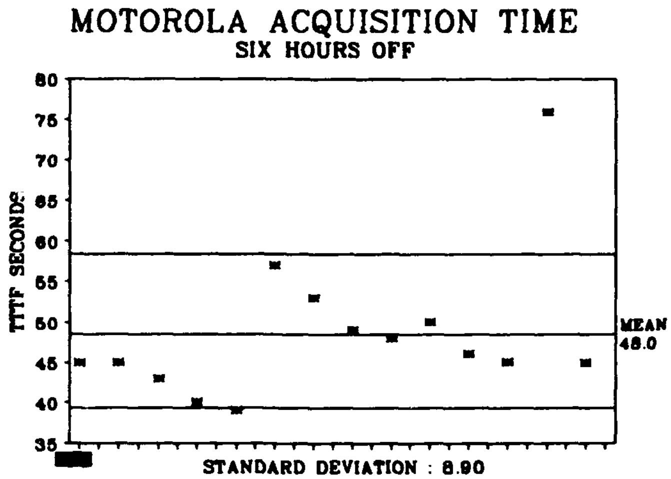


Figure F.11 : Motorola Acquisition Time After 6 Hours Off

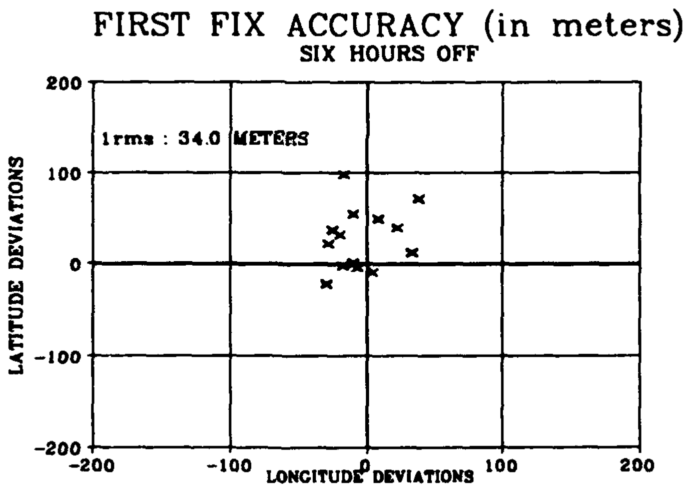


Figure F.12 : Motorola First Fix Accuracy After 6 Hours Off

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, California 93943-5000	2
Dr. Robert B. Mcghee Code CS/Mz Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	2
Dr. James R. Clynch Code OC/Cl Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	3
Dr. Se-Hung Kwak Code CS/Kw Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	3
Lcdr. Clark D. Stevens Naval Training Systems Center Orlando, Florida 32836-3224	1